C++ is all you need

Ning Zhang

2025第二届科学计算实战技术培训班, 01/18

Who am I

- **教育经历**
  - 2014-2018 学士 北京大学化学与分子工程学院
  - 2018-2023 博士 北京大学化学与分子工程学院
    - 导师 ： 刘文剑教授，肖云龙教授
    - 研究方向 ： (1) 发展新一代强关联体系计算方法 (2) 开发高性能通用计算软件
- **研究兴趣** **iCIPT2 / SOiCI, iCISO /** 4C-iCIPT2, **MetaWave**
  - 强关联体系电子结构，相对论量子化学，**通用**高性能科学计算软件**设计**与开发
- 第 9 届中国化学会理论化学优秀博士奖

MetaWave

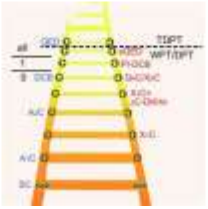
- iCIPT2
  - Near-exact quantum chemistry methods
  - Selected configuration interaction plus perturbation theory
- Feature of **MetaWave**
  - Unified implementation, fully templated (C→ **C with class** → **C++ with TMP**)
    in BDF
  - Supports various kinds of Hamiltonian
  - Supports (almost) all Abelian symmetries and spin symmetries
  - High performance, **but** easy to extend

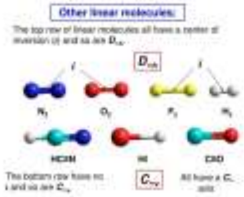
MPI-4C-iCIPT2 in **2** hours

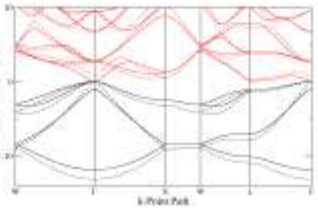
MetaWave

Non-relativistic
Relativistic
Symmetry
Cavity QED
Magnetic field
....

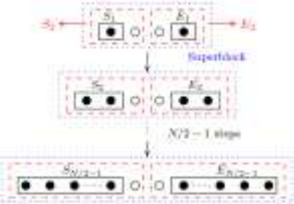
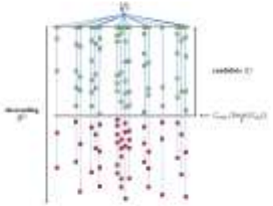
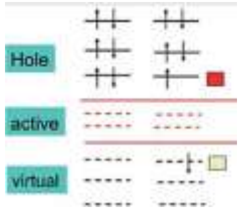
Hamiltonian

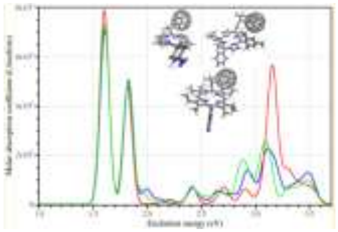






Wavefunction



Selection
MRPT
PASPT2
Spectroscopy
Fragmentation
....

C++ template metaprogramming

OpenMP

MPI

GPU

DCU

MetaWave

```
template <typename CfgSpaceTy, typename IntegralsTy>
class SelectionThreadContext : public
ThreadContext_ABC<Parallel::tag::thread_context_mode::upload_after_finish_subtask, size_t>
{
public:
    /* typedef */

    using sngl_record_t      = typename
SelectionPerturbation::__typebinder::CfgSpaceBinder<CfgSpaceTy>::sngl_record_t;
    using dbl_record_t       = typename
SelectionPerturbation::__typebinder::CfgSpaceBinder<CfgSpaceTy>::dbl_record_t;
    using hmat_calculator_t = typename
SelectionPerturbation::__typebinder::CfgSpaceBinder<CfgSpaceTy>::hmat_calculator_t;
    using ccf_getter_t       = typename
SelectionPerturbation::__typebinder::CfgSpaceBinder<CfgSpaceTy>::hmat_calculator_t;

/// …… ommited
};
```

MetaWave

	Symmetry	Det	CSF	Selection	Perturbation	Natural Orbital	RDM	CoreExt	MRPT2
Molecule D_{2h}	√	√	√	√	√	√	√	√	√
Molecule L_z	√	√	√	√	√	√	√	√	×
Solid PBC	√	√	√	√	√	√	√	√	×
Relativistic Scalar orbitals	√	--	√	√	√	×	√	×	×
Relativistic Spinor orbitals	√	--	√	√	√	√	√	√	×
Fragmentation	√	×	√	√	×	√	√	×	×

Remark

- Explain **polymorphism** (and Template Metaprogramming) with examples in **numerical linear algebra** and **quantum chemistry**
- Won't get into the technical details.
- **C++ is extremely powerful in developing high-performance generic scientific computation package**
- **(1) Remove unnecessary copy-paste;** (2) more human-readable'
- (3) **abstract syntax tree (4) how theory and code become consistent**
- Code of today's talk

[2025SciComProCoder/lec12_C++_in_Scientific_Computation at main · SciProCoder/2025SciComProCoder]()

```
mkdir build
cd build
cmake ..
make -j
```

Outline

1. Why C++ ?

2. Polymorphism in C++

3*. Advanced Examples

Outline of Polymorphism in C++

1. Overloading, Interface of Linear Algebra Ops
2. Template, Automatic Unrolling
3. Abstract Base Class, Parallel

Outline of Advanced Examples

1. Type traits

2. Concept

3. *Expression Templates

Outline

1. **Why C++ ?**

2. **Polymorphism in C++**

3*. **Advanced Examples**

Why C++ ?

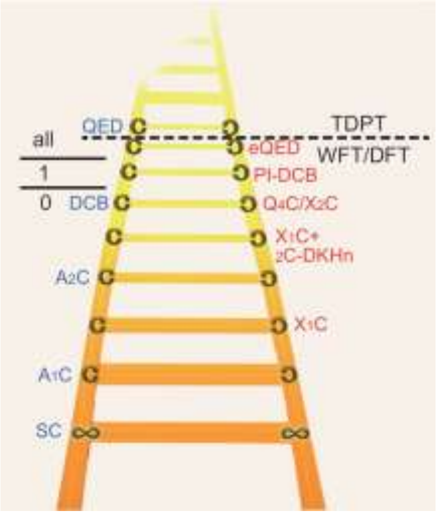
Scenario: Suppose you now need to develop software similar to **PowerPoint.** You need to design a series of functions to **print text**. It should support printing in different **fonts, colors, font sizes**, and include **features** such as underline, strikethrough, bold, italic …..

- If you use a procedural programming paradigm, how much code would you write? (**exponentially!**)

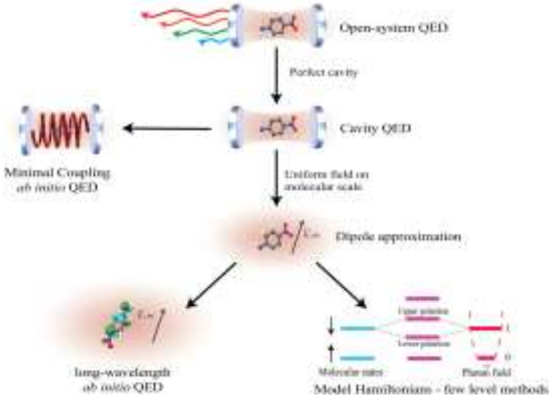
- What would you do if you need to add new fonts and new colors? (**Extendible?**)

Complexity of Quantum Chemistry Package

- Hamiltonian
  - Non-relativistic
  - Relativistic
  - QED effects
  - Electron-phonon coupling
  - Cavity QED
  - + external field
  - ....



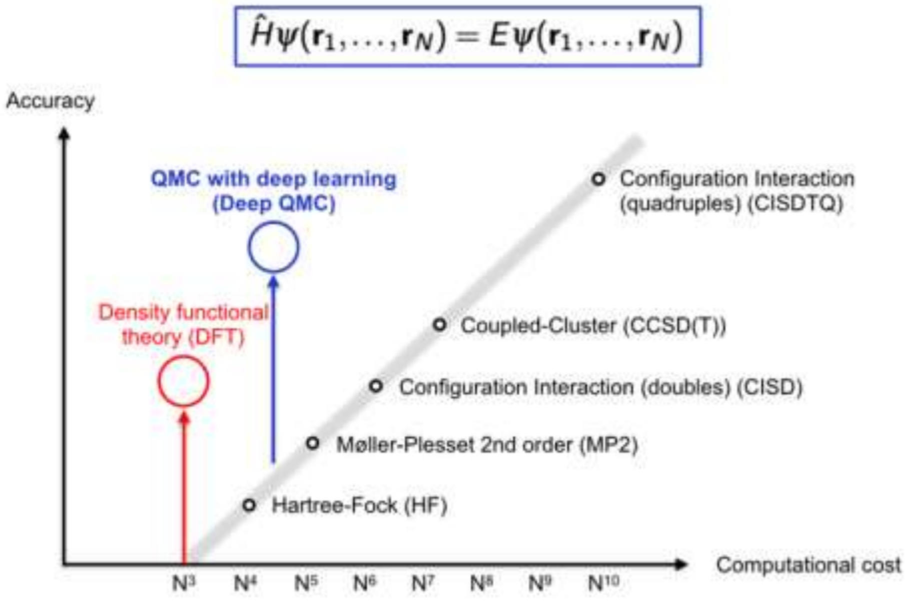
(Liu, 2010)



(Koch, et., 2023)

Complexity of Quantum Chemistry Package

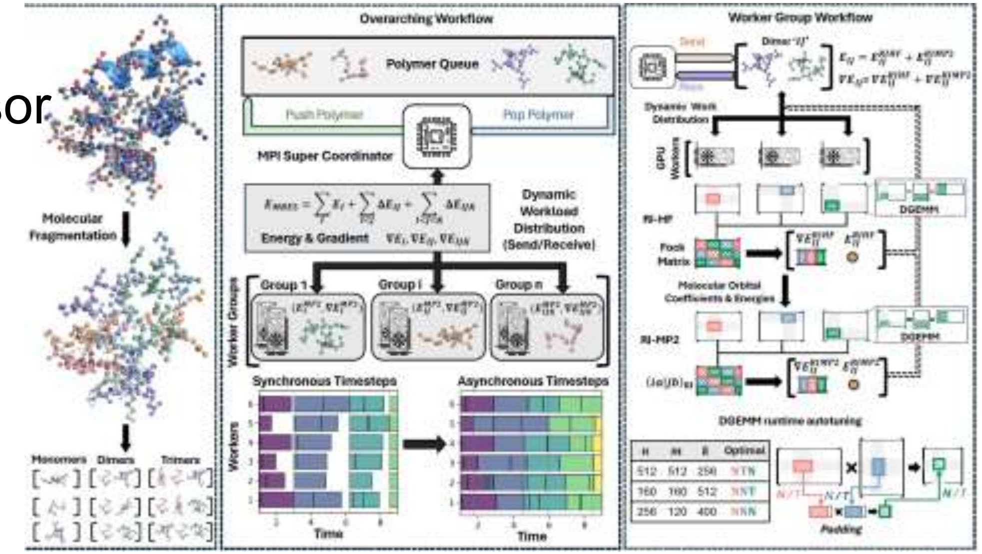
- Method
  - Mean-field
  - Perturbation theory
  - Configuration Interaction
  - Coupled Cluster
  - Quantum Monte Carlo
  - Tensor-network states (DMRG,TTNS,PEPS,MERA,…..)
  - + Linear Response
  - ………



(Extance, 2020, blog)

Complexity of Quantum Chemistry Package

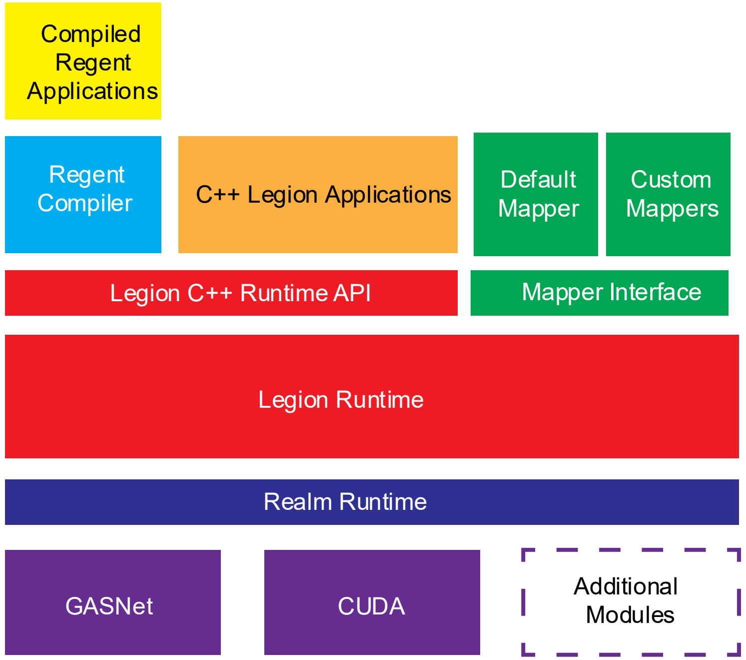
- Implementation, “infra structure”
  - Tensor
  - Block-, Sparse-, Block-sparse-,…, Tensor
  - (t)BLAS 1/2/3
  - Contraction
  - Hardware, CPU/GPU/TPU/NPU
  - Parallelization, OpenMP/MPI
  - ………..



(arXiv: 2410.21888, 2024 Gordon Bell Prize)

Complexity of Quantum Chemistry Package

Legion Programming System



Legion: Programming Distributed Heterogeneous Architectures with Logical Regions

A Hybrid Approach to Automatic Program Parallelization via Efficient Tasking with Composable Data Partitioning

Realm: Performance Portability through Composable Asynchrony

Regent: A High-Productivity Programming Language for Implicit Parallelism with Logical Regions

Scaling Implicit Parallelism with Index Launches

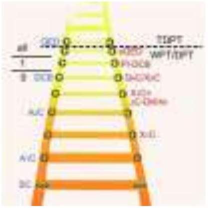
Automated Discovery of Machine Learning Optimizations

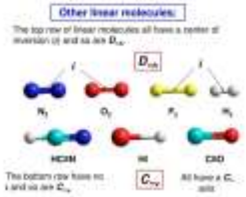
Complexity of Quantum Chemistry Package

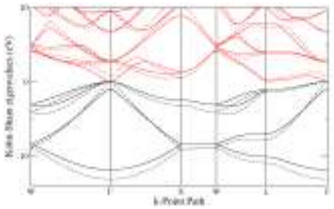
SHANDONG UNIVERSITY

Non-relativistic
Relativistic
Symmetry
Cavity QED
Magnetic field
…….

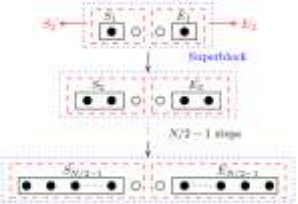
Hamiltonian

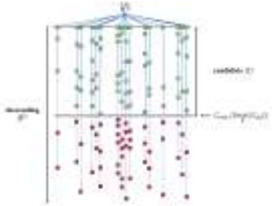


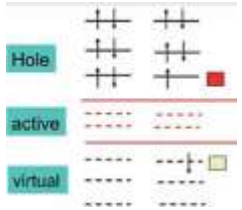


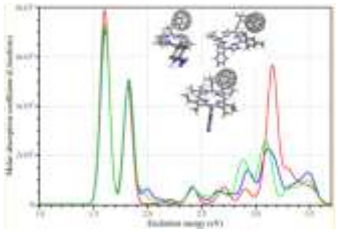


Wavefunction









Selection
MRPT
PASPT2
Spectroscopy
Fragmentation
…….

C++ template metaprogramming

OpenMP | MPI | GPU | DCU

Complexity of Quantum Chemistry Package

- Quantum Chemistry Package

  = (Hamiltonian * Method) * Implementation (data structure / parallel)

- **Large “Entropy”**

Complexity of Quantum Chemistry Package

- Quantum Chemistry Package

  = (Hamiltonian * Method) * Implementation (data structure / parallel)

- **Large “Entropy”**
  - Macro level
  - Micro level
  - Too much copy/paste

Complexity of Quantum Chemistry Package

$$\langle \tilde{\mathbf{d}} | E_{ij} | \mathbf{d} \rangle = \Pi_{r\in\Omega_1} W_r \cdot \Pi_{r\notin\Omega_1} \delta_{\tilde{d}_r, d_r}$$

$$= \Pi_{r\in\Omega_1} W(Q_r; \tilde{d}_r d_r, \Delta b_r, b_r) \cdot \Pi_{r\notin\Omega_1} \delta_{\tilde{d}_r, d_r}, \quad \Delta b_r = b_r - \tilde{b}_r,$$

$$\langle \tilde{\mathbf{d}} | e_{ij,kl} | \mathbf{d} \rangle = \Pi_{r\in\Omega_2^n} W_r \left[\sum_{X=0}^{1} \Pi_{s\in\Omega_2^o} W_s^{(1)}(X) \right] \cdot \Pi_{r\notin\Omega_2} \delta_{\tilde{d}_r, d_r}$$

$$= \Pi_{r\in\Omega_2^n} W(Q_r; \tilde{d}_r d_r, \Delta b_r, b_r) \left[\Pi_{s\in\Omega_2^o} W_s^{(1)}(Q_s; \tilde{d}_s d_s, \Delta b_s = 0, X = 0) \right.$$

$$\left. + \Pi_{s\in\Omega_2^o} W_s^{(1)}(Q_s; \tilde{d}_s d_s, \Delta b_s, X = 1) \right] \cdot \Pi_{r\notin\Omega_2} \delta_{\tilde{d}_r, d_r}.$$

- Qr: 26 种, d : 4 种, △b : 5 种, b 是整数

Complexity of Quantum Chemistry Package

表 A.4　B_R、B^R、B_L 和 B^L 型片断因子的值。$\Delta b = b - \tilde{b}$。

$\tilde{d}d$	$W(B_R;\tilde{d}d,\Delta b,b;X)$				$W(B^L;\tilde{d}d,\Delta b,b;X)$			
	$X=0$		$X=1$		$X=0$	$X=1$		
	$\Delta b=-1$	$\Delta b=1$	$\Delta b=-1$	$\Delta b=1$	$\Delta b=0$	$\Delta b=-2$	$\Delta b=0$	$\Delta b=2$
01	-	$-\sqrt{\frac{b+1}{2b}}$	-1	$-\sqrt{\frac{b-1}{2b}}$	$1/\sqrt{2}$	-	$-\sqrt{\frac{b+2}{2b}}$	$-\sqrt{\frac{b+1}{b}}$
02	$\sqrt{\frac{b+1}{2(b+2)}}$	-	$-\sqrt{\frac{b+3}{2(b+2)}}$	-1	$1/\sqrt{2}$	$\sqrt{\frac{b+1}{b+2}}$	$\sqrt{\frac{b}{2(b+2)}}$	-
13	$-1/\sqrt{2}$	-	$\sqrt{\frac{b}{2(b+2)}}$	$\sqrt{\frac{b-1}{2b}}$	$\sqrt{\frac{b}{2(b+1)}}$	1	$\sqrt{\frac{b+2}{2(b+1)}}$	-
23	-	$-1/\sqrt{2}$	$-\sqrt{\frac{b+3}{b+2}}$	$-\sqrt{\frac{b+2}{2b}}$	$-\sqrt{\frac{b+2}{2(b+1)}}$	-	$\sqrt{\frac{b}{2(b+1)}}$	1
$\tilde{d}d$	$W(B_L;\tilde{d}d,\Delta b,b;X)$				$W(B^R;\tilde{d}d,\Delta b,b;X)$			
	$X=0$		$X=1$		$X=0$	$X=1$		
	$\Delta b=-1$	$\Delta b=1$	$\Delta b=-1$	$\Delta b=1$	$\Delta b=0$	$\Delta b=-2$	$\Delta b=0$	$\Delta b=2$
10	$-1/\sqrt{2}$	-	$-\sqrt{\frac{b}{2(b+2)}}$	$-\sqrt{\frac{b-1}{b}}$	$\sqrt{\frac{b}{2(b+1)}}$	-1	$-\sqrt{\frac{b+2}{2(b+1)}}$	-
20	-	$-1/\sqrt{2}$	$\sqrt{\frac{b+3}{b+2}}$	$\sqrt{\frac{b+2}{2b}}$	$-\sqrt{\frac{b+2}{2(b+1)}}$	-	$-\sqrt{\frac{b}{2(b+1)}}$	-1
31	-	$\sqrt{\frac{b+1}{2b}}$	-1	$-\sqrt{\frac{b-1}{2b}}$	$-1\sqrt{2}$	-	$-\sqrt{\frac{b+2}{2b}}$	$-\sqrt{\frac{b+1}{b}}$
32	$-\sqrt{\frac{b+1}{2(b+2)}}$	-	$-\sqrt{\frac{b+3}{2(b+2)}}$	-1	$-1\sqrt{2}$	$\sqrt{\frac{b+1}{b+2}}$	$\sqrt{\frac{b}{2(b+2)}}$	-

Complexity of Quantum Chemistry Package

- Quantum Chemistry Package

  = (Hamiltonian * Method) * Implementation

- **Large "Entropy"**
  - Macro level
  - Micro level (dgemm, sparse matrix, molecular integrals, UGA, ….)
  - Too much copy/paste or cannot be extendible

Complexity of Quantum Chemistry Package

- Quantum Chemistry Package

  = (Hamiltonian * Method) * Implementation

- **Large “Entropy”**
- **Gap** between Code and Formulas, not human-readable
  - matrix matrix multiplication v.s. triple for loop
  - CCSD v.s. ?
- **Decouple** different aspects (as much as possible)

分子/固体+电子/声子/质子 +
相对论效应 + 响应性质

Complexity of Quantum Chemistry Package

[pyscf/pyscf/cc/rintermediates.py at master · pyscf/pyscf]

```python
def cc_Foo(t1, t2, eris):
    nocc, nvir = t1.shape
    foo = eris.fock[:nocc,:nocc]
    eris_ovov = np.asarray(eris.ovov)
    Fki  = 2*lib.einsum('kcld,ilcd->ki', eris_ovov, t2)
    Fki -=   lib.einsum('kdlc,ilcd->ki', eris_ovov, t2)
    Fki += 2*lib.einsum('kcld,ic,ld->ki', eris_ovov, t1, t1)
    Fki -=   lib.einsum('kdlc,ic,ld->ki', eris_ovov, t1, t1)
    Fki += foo
    return Fki

def cc_Fvv(t1, t2, eris):
    nocc, nvir = t1.shape
    fvv = eris.fock[nocc:,nocc:]
    eris_ovov = np.asarray(eris.ovov)
    Fac  =-2*lib.einsum('kcld,klad->ac', eris_ovov, t2)
    Fac +=   lib.einsum('kdlc,klad->ac', eris_ovov, t2)
    Fac -= 2*lib.einsum('kcld,ka,ld->ac', eris_ovov, t1, t1)
    Fac +=   lib.einsum('kdlc,ka,ld->ac', eris_ovov, t1, t1)
    Fac += fvv
    return Fac

def cc_Fov(t1, t2, eris):
    nocc, nvir = t1.shape
    fov = eris.fock[:nocc,nocc:]
    eris_ovov = np.asarray(eris.ovov)
    Fkc  = 2*np.einsum('kcld,ld->kc', eris_ovov, t1)
    Fkc -=   np.einsum('kdlc,ld->kc', eris_ovov, t1)
    Fkc += fov
    return Fkc
```

```python
def Loo(t1, t2, eris):
    nocc, nvir = t1.shape
    fov = eris.fock[:nocc,nocc:]
    Lki = cc_Foo(t1, t2, eris) + np.einsum('kc,ic->ki',fov, t1)
    eris_ovoo = np.asarray(eris.ovoo)
    Lki += 2*np.einsum('lcki,lc->ki', eris_ovoo, t1)
    Lki -=   np.einsum('kcli,lc->ki', eris_ovoo, t1)
    return Lki

def Lvv(t1, t2, eris):
    nocc, nvir = t1.shape
    fov = eris.fock[:nocc,nocc:]
    Lac = cc_Fvv(t1, t2, eris) - np.einsum('kc,ka->ac',fov, t1)
    eris_ovvv = np.asarray(eris.get_ovvv())
    Lac += 2*np.einsum('kdac,kd->ac', eris_ovvv, t1)
    Lac -=   np.einsum('kcad,kd->ac', eris_ovvv, t1)
    return Lac
```

Complexity of Quantum Chemistry Package

pyscf/pyscf/cc/rintermediates.py at master · pyscf/pyscf

```python
def update_amps(cc, t1, t2, eris):
    # Ref: Hirata et al., J. Chem. Phys. 120, 2581 (2004) Eqs.(35)-(36)
    nocc, nvir = t1.shape
    fock = eris.fock

    fov = fock[:nocc,nocc:].copy()
    foo = fock[:nocc,:nocc].copy()
    fvv = fock[nocc:,nocc:].copy()

    Foo = imd.cc_Foo(t1,t2,eris)
    Fvv = imd.cc_Fvv(t1,t2,eris)
    Fov = imd.cc_Fov(t1,t2,eris)

    # Move energy terms to the other side
    Foo -= np.diag(np.diag(foo))
    Fvv -= np.diag(np.diag(fvv))
```

```python
    # T1 equation
    t1new = np.asarray(fov).conj().copy()
    t1new +=-2*einsum('kc,ka,ic->ia', fov, t1, t1)
    t1new +=   einsum('ac,ic->ia', Fvv, t1)
    t1new +=  -einsum('ki,ka->ia', Foo, t1)
    t1new += 2*einsum('kc,kica->ia', Fov, t2)
    t1new +=  -einsum('kc,ikca->ia', Fov, t2)
    t1new +=   einsum('kc,ic,ka->ia', Fov, t1, t1)
    t1new += 2*einsum('kcai,kc->ia', eris.ovvo, t1)
    t1new +=  -einsum('kiac,kc->ia', eris.oovv, t1)
    eris_ovvv = np.asarray(eris.ovvv)
    t1new += 2*einsum('kdac,ikcd->ia', eris_ovvv, t2)
    t1new +=  -einsum('kcad,ikcd->ia', eris_ovvv, t2)
    t1new += 2*einsum('kdac,kd,ic->ia', eris_ovvv, t1, t1)
    t1new +=  -einsum('kcad,kd,ic->ia', eris_ovvv, t1, t1)
    t1new +=-2*einsum('kilc,klac->ia', eris.ooov, t2)
    t1new +=   einsum('likc,klac->ia', eris.ooov, t2)
    t1new +=-2*einsum('kilc,lc,ka->ia', eris.ooov, t1, t1)
    t1new +=   einsum('likc,lc,ka->ia', eris.ooov, t1, t1)
```

Complexity of Quantum Chemistry Package

[pyscf/pyscf/cc/rintermediates.py at master · pyscf/pyscf]()

```python
# T2 equation
t2new = np.asarray(eris.ovov).conj().transpose(0,2,1,3).copy()
if cc.cc2:
    Woooo2 = np.asarray(eris.oooo).transpose(0,2,1,3).copy()
    Woooo2 += einsum('kilc,jc->klij', eris.ooov, t1)
    Woooo2 += einsum('ljkc,ic->klij', eris.ooov, t1)
    Woooo2 += einsum('kcld,ic,jd->klij', eris.ovov, t1, t1)
    t2new += einsum('klij,ka,lb->ijab', Woooo2, t1, t1)
    Wvvvv = einsum('kcbd,ka->abcd', eris_ovvv, -t1)
    Wvvvv = Wvvvv + Wvvvv.transpose(1,0,3,2)
    Wvvvv += np.asarray(eris.vvvv).transpose(0,2,1,3)
    t2new += einsum('abcd,ic,jd->ijab', Wvvvv, t1, t1)
    Lvv2 = fvv - einsum('kc,ka->ac', fov, t1)
    Lvv2 -= np.diag(np.diag(fvv))
    tmp = einsum('ac,ijcb->ijab', Lvv2, t2)
    t2new += (tmp + tmp.transpose(1,0,3,2))
    Loo2 = foo + einsum('kc,ic->ki', fov, t1)
    Loo2 -= np.diag(np.diag(foo))
    tmp = einsum('ki,kjab->ijab', Loo2, t2)
    t2new -= (tmp + tmp.transpose(1,0,3,2))
```

```python
else:
    Loo = imd.Loo(t1, t2, eris)
    Lvv = imd.Lvv(t1, t2, eris)
    Loo -= np.diag(np.diag(foo))
    Lvv -= np.diag(np.diag(fvv))
    Woooo = imd.cc_Woooo(t1, t2, eris)
    Wvoov = imd.cc_Wvoov(t1, t2, eris)
    Wvovo = imd.cc_Wvovo(t1, t2, eris)
    Wvvvv = imd.cc_Wvvvv(t1, t2, eris)
    tau = t2 + einsum('ia,jb->ijab', t1, t1)
    t2new += einsum('klij,klab->ijab', Woooo, tau)
    t2new += einsum('abcd,ijcd->ijab', Wvvvv, tau)
    tmp = einsum('ac,ijcb->ijab', Lvv, t2)
    t2new += (tmp + tmp.transpose(1,0,3,2))
    tmp = einsum('ki,kjab->ijab', Loo, t2)
    t2new -= (tmp + tmp.transpose(1,0,3,2))
    tmp  = 2*einsum('akic,kjcb->ijab', Wvoov, t2)
    tmp -=   einsum('akci,kjcb->ijab', Wvovo, t2)
    t2new += (tmp + tmp.transpose(1,0,3,2))
    tmp = einsum('akic,kjbc->ijab', Wvoov, t2)
    t2new -= (tmp + tmp.transpose(1,0,3,2))
    tmp = einsum('bkci,kjac->ijab', Wvovo, t2)
    t2new -= (tmp + tmp.transpose(1,0,3,2))
```

Complexity of Quantum Chemistry Package

[pyscf/pyscf/cc/rintermediates.py at master · pyscf/pyscf]

```
tmp2  = einsum('kibc,ka->abic', eris.oovv, -t1)
tmp2 += np.asarray(eris.ovvv).conj().transpose(1,3,0,2)
tmp = einsum('abic,jc->ijab', tmp2, t1)
t2new += (tmp + tmp.transpose(1,0,3,2))
tmp2  = einsum('kcai,jc->akij', eris.ovvo, t1)
tmp2 += np.asarray(eris.ooov).transpose(3,1,2,0).conj()
tmp = einsum('akij,kb->ijab', tmp2, t1)
t2new -= (tmp + tmp.transpose(1,0,3,2))

mo_e = eris.fock.diagonal().real
eia = mo_e[:nocc,None] - mo_e[None,nocc:]
eijab = lib.direct_sum('ia,jb->ijab',eia,eia)
t1new /= eia
t2new /= eijab
```

Why C++ ?

C++ as a Federation of Languages (Effective C++, Item 1)

- C

  Blocks, statements, the preprocessor, built-in data types, arrays, pointers, etc.

Why C++ ?

C++ as a Federation of Languages (Effective C++, Item 1)

- C

  Blocks, statements, the preprocessor, built-in data types, arrays, pointers, etc.

- Object-Oriented C++

  Classes, encapsulation, inheritance, polymorphism, virtual functions, etc.

Why C++ ?

C++ as a Federation of Languages (Effective C++, Item 1)

- C

  Blocks, statements, the preprocessor, built-in data types, arrays, pointers, etc.

- Object-Oriented C++

  Classes, encapsulation, inheritance, polymorphism, virtual functions, etc.

- **Template C++ , Generic C++**

  Template metaprogramming (TMP)

Why C++ ?

C++ as a Federation of Languages (Effective C++, Item 1)

- C

  Blocks, statements, the preprocessor, built-in data types, arrays, pointers, etc.

- Object-Oriented C++

  Classes, encapsulation, inheritance, polymorphism, virtual functions, etc.

- **Template C++ , Generic C++**

  Template metaprogramming (TMP)

- **The STL**

  Template library

Why C++ ?

C++ as a Federation of Languages (Effective C++, Item 1)

- C

  Blocks, statements, the preprocessor, built-in data types, arrays, pointers, etc.

- Object-Oriented C++

  Classes, encapsulation, inheritance, polymorphism, virtual functions, etc.

- **Template C++ , Generic C++**

  Template metaprogramming (TMP)

- **The STL**

  Template library

Why not C / Fortran ?

Why C++ ?

C++ as a Federation of Languages (Effective C++, Item 1)

- C

  Blocks, statements, the preprocessor, built-in data types, arrays, pointers, etc.

- Object-Oriented C++

  Classes, encapsulation, inheritance, polymorphism, virtual functions, etc.

- **Template C++ , Generic C++**

  Template metaprogramming (TMP)

- **The STL**

  Template library

Why not C / Fortran ?

Why not python?

Why C++ ?

C++ as a Federation of Languages (Effective C++, Item 1)

- C

    Blocks, statements, the preprocessor, built-in data types, arrays, pointers, etc.

- Object-Oriented C++

    Classes, encapsulation, inheritance, polymorphism, virtual functions, etc.

- **Template C++ , Generic C++**

    Template metaprogramming (TMP)

- **The STL**

    Template library

Why not C / Fortran ?

Why not python?

Why C++!

Why C++ ?

- How to implement $y = ax + y$ in C / Fortran with the same name?

  C impossible, Fortran interface

```
cblas_saxpy(n,  a, x, incx, y, incy);
cblas_daxpy(n,  a, x, incx, y, incy);
cblas_caxpy(n, &a, x, incx, y, incy);
cblas_zaxpy(n, &a, x, incx, y, incy);

y = a * x + y;
y = x * a + y;
y+= a * x;
y+= x * a;
```

more human-readable

Why C++ ? – Polymorphism

- **Polymorphism** is the provision of a single interface to entities of different types or the use of a single symbol to represent multiple different types.

https://en.wikipedia.org/wiki/Polymorphism_(computer_science)

- For F(x,y), apply F on x,y (even on different sets of params) whenever it is possible.

  F can be any operator/function/……

- **Purpose**: Avoid or even remove duplication of effort.

```
axpy(n, a, x, incx, y, incy);

axpy(a, vecx, vecy);

y = a * x + y

y+= a * x
```

Hartree Fock 优化算法与哈密顿量无关

不同的哈密顿量应该用同一套优化算法的代码

Why C++ ? – Polymorphism

- **Polymorphism** is the provision of a single interface to entities of different types or the use of a single symbol to represent multiple different types.

https://en.wikipedia.org/wiki/Polymorphism_(computer_science)

- **Ad hoc polymorphism**: defines a common interface for an arbitrary set of individually specific types. (**Function overloading, operator overloading**)

- **Parametric polymorphism**: not specifying concrete types and instead use abstract symbols that can substitute for any type (**Template**)

```
template <typename T>
T operator+(const T &a, const T &b);
```

Why C++ ? – Polymorphism

- **Polymorphism** is the provision of a single interface to entities of different types or the use of a single symbol to represent multiple different types.

https://en.wikipedia.org/wiki/Polymorphism_(computer_science)

- **Ad hoc polymorphism**: defines a common interface for an arbitrary set of individually specific types. (**Function overloading, operator overloading**)

- **Parametric polymorphism**: not specifying concrete types and instead use abstract symbols that can substitute for any type (**Template**)

- **Subtyping**: when a name denotes instances of many different classes related by some common superclass (**Abstract base class**, **virtual function**)

Why C++ ? – Polymorphism

- **Polymorphism** is the provision of a single interface to entities of different types or the use of a single symbol to represent multiple different types.

https://en.wikipedia.org/wiki/Polymorphism_(computer_science)

- **Static polymorphism**

In computing, static dispatch is a form of polymorphism **fully resolved during compile time**.

https://en.wikipedia.org/wiki/Static_dispatch

- **Dynamic polymorphism**

In computer science, dynamic dispatch is the process of selecting which implementation of a polymorphic operation (method or function) to call **at run time**.

https://en.wikipedia.org/wiki/Dynamic_dispatch

Why C++ ?

- 模块**解耦** (哈密顿量，量子化学方法，并行, 线性代数操作)
- **可以用更少的代码实现更多的功能, 表达力强** (软件包 = 简单功能 * 排列组合/复合, 后者尽可能自动化)

Why C++ ?

- 模块**解耦**　　(哈密顿量，量子化学方法，并行, 线性代数操作)
- **可以用更少的代码实现更多的功能, 表达力强**　(软件包 = 简单功能 * 排列组合/复合, 后者尽可能自动化)

牛肉 = 牛 + 肉，　beaf != bull + meat , pork != pig + meat

葡萄 grape 葡萄干 raisin 葡萄酒 wine

Pneumonoultramicroscopicsilicovolcanoconiosis 尘肺病

有机化学命名，分类学科，属命名

Why C++ ?

- 模块**解耦** (哈密顿量，量子化学方法，并行, 线性代数操作)
- **可以用更少的代码实现更多的功能, 表达力强** (软件包 = 简单功能 * 排列组合/复合, 后者尽可能自动化)
- 维护成本低
- **编译器静态检查** (static_assert, concept)
- 出错率较低 (编译器辅助检查， 编译器执行 Ctrl + C, Ctrl + V)
- ....

Why C++ ?

- 模块**解耦**　　(哈密顿量，量子化学方法，并行, 线性代数操作)
- **可以用更少的代码实现更多的功能, 表达力强**　(软件包 = 简单功能 * 排列组合/复合, 后者尽可能自动化)
- 维护成本低
- **编译器静态检查** (static_assert, concept)
- 出错率较低 (编译器辅助检查，编译器执行 Ctrl + C, Ctrl + V)
- ………..

代价是什么?

Why not C++ ?

- 语法过于**复杂**，学习成本**高**, **no free lunch**
  - C / Fortran : 表达力**弱**，性能强，学习成本低
  - Python : 表达力**中**，性能**中**，学习成本**中**
  - C++ : 表达力强，性能强，学习成本**高**
- 不适用结构简单目标单一的计算软件的开发
- 对开发，维护人员要求**高**
- **“1000个读者眼中就会有1000个哈姆雷特”**
- ………..

Outline

1. Why C++ ?

2. Polymorphism in C++

3*. Advanced Examples

Outline of Polymorphism in C++

1. Overloading, Interface of Linear Algebra Ops

2. Template, Automatic Unrolling

3. Abstract Base Class, Parallel

Outline of Advanced Examples

1. Type traits

2. Concept

3. *Expression Templates

Outline of Polymorphism in C++

1. **Overloading, Interface of Linear Algebra Ops**

2. Template, Automatic Unrolling

3. Abstract Base Class, Parallel

Function Overloading

- In some programming languages, function overloading or method overloading is the ability to create multiple functions of the **same name** with **different implementations**. Calls to an overloaded function will run a specific implementation of that function appropriate to the context of the call, allowing one function call to perform different tasks depending on context.

https://en.wikipedia.org/wiki/Function_overloading

- 编译器根据**输入参数**匹配同一个函数的不同版本

- In C++, function overloading happens at **compile** time. (**static**)

Function Overloading

```
cblas_saxpy(n,  a, x, incx, y, incy);

cblas_daxpy(n,  a, x, incx, y, incy);

cblas_caxpy(n, &a, x, incx, y, incy);

cblas_zaxpy(n, &a, x, incx, y, incy);
```

标量和函数名绑定，无法统一处理！

Function Overloading

```
inline void AXPY(const MKL_INT n, const float a,
const float *x, const MKL_INT incx, float *y, const MKL_INT incy)
{
    cblas_saxpy(n, a, x, incx, y, incy);
}

inline void AXPY(const MKL_INT n, const double a,
const double *x, const MKL_INT incx, double *y, const MKL_INT incy)
{
    cblas_daxpy(n, a, x, incx, y, incy);
}

inline void AXPY(const MKL_INT n, const complex_double a,
const complex_double *x, const MKL_INT incx, complex_double *y, const MKL_INT incy)
{
    cblas_zaxpy(n, (const void *)&a, x, incx, y, incy);
}
```

编译器通过输入变量类型自动匹配正确的函数

Interface of Linear Algebra Interface

- **线性代数操作与标量无关**（最底层的多态)
- **隔离线性代数操作的实现和上层的调用**

Advanced Interface

```cpp
struct dot_impl
{
    /*!
     * \brief Apply the functor to a and b
     * \param a the left hand side
     * \param b the left hand side
     * \return the dot product of a and b
     */
    template <typename A, typename B>
    static value_t<A> apply(const A& a, const B& b)
    {
        constexpr auto impl = select_dot_impl<A, B>::value;

        if constexpr (impl == dot_impl::BLAS)
        {
            return Backend::blas::dot(a, b);
        }
        else if constexpr (impl == dot_impl::CUBLAS)
        {
            return Backend::cublas::dot(a, b);
        }
        else if constexpr (impl == dot_impl::VEC)
        {
            return Backend::vec::dot(a, b);
        }
        else
        {
            return Backend::standard::dot(a, b);
        }
    }
};
```

- A, B lhs, rhs 的类型
- value_t<T> 返回 T 的标量类型
- select_dot_impl<A,B> 可以根据 A, B

类型种包含的信息在编译器决定调用哪个实现

- If constexpr 编译期就可以判断走哪个分支

Interface of Linear Algebra Interface

- **线性代数操作与标量无关**（最底层的多态）
- **隔离线性代数操作的实现和上层的调用**
- **可移植性**
- **性能测试**

Exercise: exercise/01-linear_algebra_interface.h

Exercise 1

Exercise: exercise/01-linear_algebra_interface.h

Answer: answer/01-linear_algebra_interface.h

```
mkdir build
cd build
cmake ..
make -j
```

```
# exercise and answer #

# add_subdirectory(exercise)
add_subdirectory(answer)
```

```
CMakeCache.txt  Makefile  cmake_install.cmake  expression_template  poly_function_overloading  poly_specialization_basic   poly_template
CMakeFiles      answer    concept              poly_ABC             poly_operator_overloading  poly_specialization_unroll  type_traits
```

Operator Overloading

- In computer programming, operator overloading, sometimes termed operator ad hoc polymorphism, is a specific case of polymorphism, where **different operators** have different implementations **depending on their arguments**. Operator overloading is generally defined by a programming language, a programmer, or both.

- Operator : +, - , *, /, +=,-=,*=,/=, %, >>, <<, &&,||, …..

- More human-readable

Operator Overloading

z = a * x + b * y

```
void multiply_add_c(const std::vector<double>& x,
const std::vector<double>& y, std::vector<double>& z,
double a, double b)
{
    size_t n = x.size();
    for (size_t i = 0; i < n; ++i)
    {
        z[i] = a * x[i] + b * y[i];
    }
}
```

What if z = a * x + b * y + c * z? 简单操作的**复合**

Operator Overloading

$z = a * x + b * y$

```
std::vector<double> operator*(const std::vector<double>& vec, double scalar)
{
    std::vector<double> result(vec.size());
    for (size_t i = 0; i < vec.size(); ++i)
    {
        result[i] = vec[i] * scalar;
    }
    return result;
}
```

Operator Overloading

$z = a * x + b * y,$

```
std::vector<double> operator*(const std::vector<double>& vec, double scalar);      (1)
std::vector<double> operator*(double scalar, const std::vector<double>& vec);      (2)
std::vector<double> operator+(const std::vector<double>& lhs, const                (3)
std::vector<double>& rhs);
```

```
z = a * x + b * y;
```

Operator Overloading

z = a * x + b * y,

```
std::vector<double> operator*(const std::vector<double>& vec, double scalar);      (1)
std::vector<double> operator*(double scalar, const std::vector<double>& vec);      (2)
std::vector<double> operator+(const std::vector<double>& lhs, const                (3)
std::vector<double>& rhs);
```

```
z = [a * x] + [b * y];    operator (2)
```

Operator Overloading

$z = a * x + b * y,$

```
std::vector<double> operator*(const std::vector<double>& vec, double scalar);     (1)
std::vector<double> operator*(double scalar, const std::vector<double>& vec);     (2)
std::vector<double> operator+(const std::vector<double>& lhs, const               (3)
std::vector<double>& rhs);
```

```
z = a * x + b * y;      operator (2)

z = t1 + t2;            operator (3)
```

Operator Overloading

$z = a * x + b * y,$

```
std::vector<double> operator*(const std::vector<double>& vec, double scalar);     (1)
std::vector<double> operator*(double scalar, const std::vector<double>& vec);     (2)
std::vector<double> operator+(const std::vector<double>& lhs, const                (3)
std::vector<double>& rhs);
```

```
z = a * x + b * y;      operator (2)

z = t1 + t2;            operator (3)
```

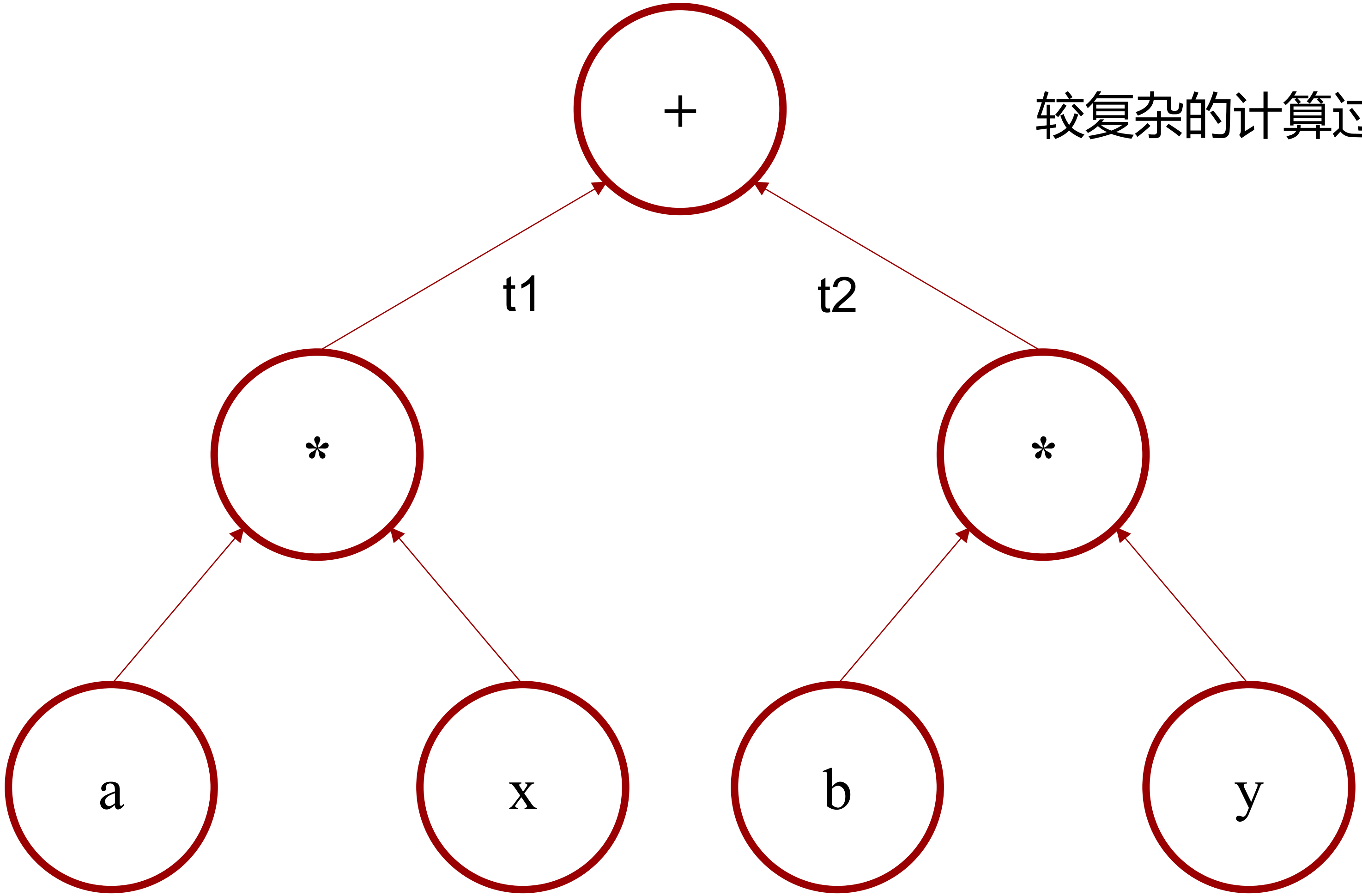
Operator Overloading

$$z = a * x + b * y,$$

```
std::vector<double> operator*(const std::vector<double>& vec, double scalar);     (1)
std::vector<double> operator*(double scalar, const std::vector<double>& vec);     (2)
std::vector<double> operator+(const std::vector<double>& lhs, const               (3)
std::vector<double>& rhs);
```

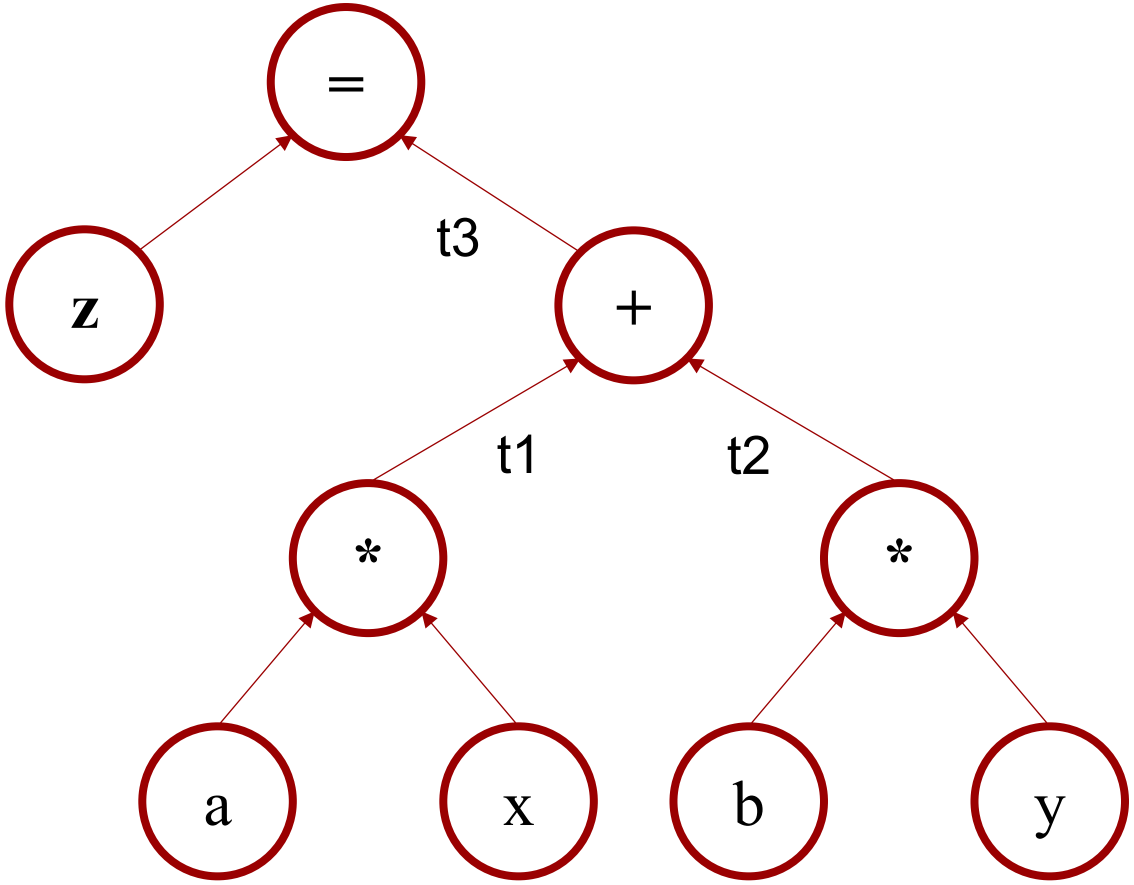
```
z = a * x + b * y;      operator (2)

z = t1 + t2;            operator (3)

z = t3;                 operator =
```

Abstract Syntax Tree



较复杂的计算过程是基本运算的复合

Abstract Syntax Tree



Operator Overloading

z = a * x + b * y

```
std::vector<double> operator*(const std::vector<double>& vec, double scalar);
std::vector<double> operator*(double scalar, const std::vector<double>& vec);
std::vector<double> operator+(const std::vector<double>& lhs, const
std::vector<double>& rhs);

const size_t         N = 1'000'000;

z = a * x + b * y;
```

7.6 ms v.s. 51.2 ms

有性能问题，中间变量需要额外申请内存

01-poly-operator_overloading.cpp

Operator Overloading

z = a * x + b * y

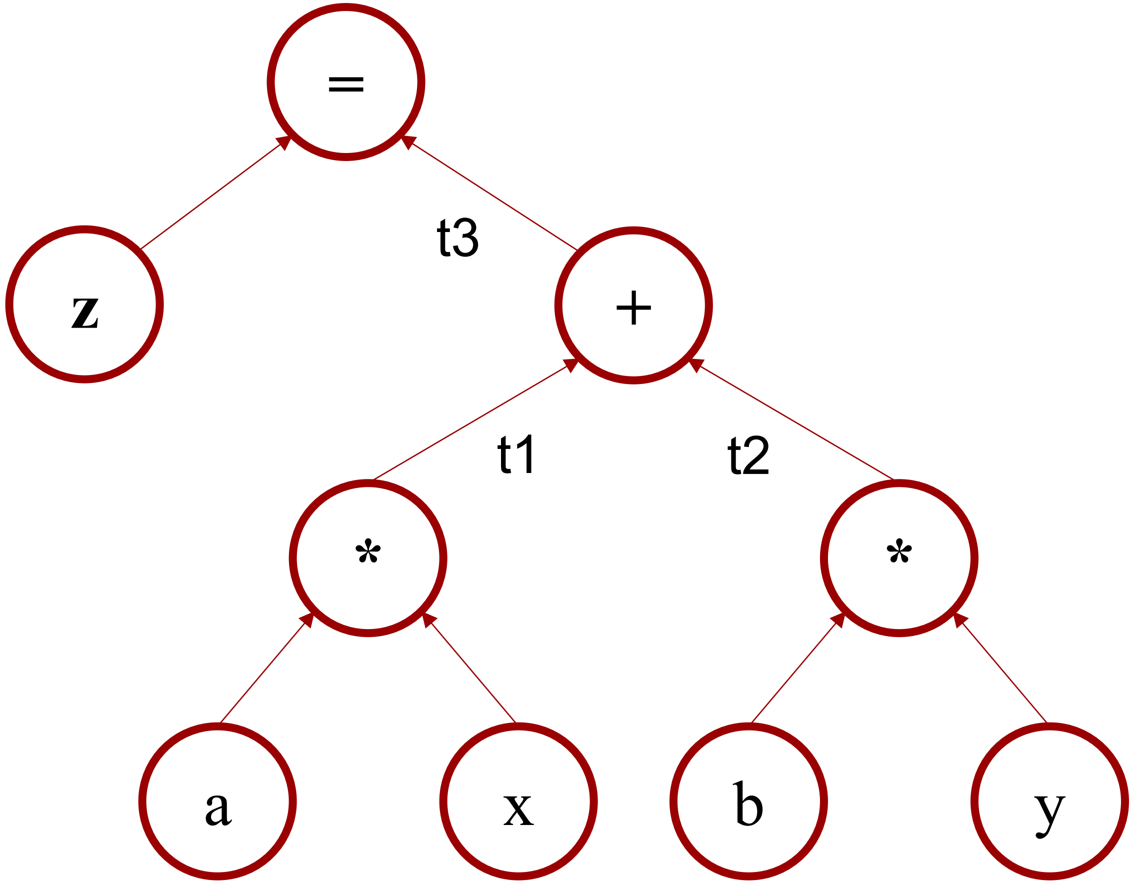
```
std::vector<double> operator*(const std::vector<double>& vec, double scalar)
{
    std::vector<double> result(vec.size());
    for (size_t i = 0; i < vec.size(); ++i)
    {
        result[i] = vec[i] * scalar;
    }
    return result;
}
```

8.95 ms v.s. 59.5 ms v.s 10.1 ms

Expression Template !

06-expression_template.cpp

Abstract Syntax Tree



Operator Overloading

z = a * x + b * y

```
void multiply_add_c(const std::vector<double>& x,
const std::vector<double>& y, std::vector<double>& z,
double a, double b)
{
    size_t n = x.size();
    for (size_t i = 0; i < n; ++i)
    {
        z[i] = a * x[i] + b * y[i];
    }
}
```

Operator Overloading

- In computer programming, operator overloading, sometimes termed operator ad hoc polymorphism, is a specific case of polymorphism, where **different operators** have different implementations **depending on their arguments**. Operator overloading is generally defined by a programming language, a programmer, or both.

- Operator : +, - , *, /, %, >>, <<, &&,||, …..

- More human-readable

- Operator overloading has often been criticized because it allows programmers to reassign the semantics of operators depending on the types of their operands.

不要滥用

不要进行防御性编程：例如 -/+ 互换, +/* 互换可能出错

Outline of Polymorphism in C++

1. Overloading, Interface of Linear Algebra Ops
2. **Template, Automatic Unrolling**
3. Abstract Base Class, Parallel

Template

- **Template** https://en.wikipedia.org/wiki/Template_(C%2B%2B)

In plain terms, a templated class or function would be the equivalent of (before "compiling") **copying and pasting the templated block** of code where it is used, and then **replacing the template parameter** with the actual one.

- **Types of Templates**
  - Function templates
  - Abbreviated function templates (since C++20)
  - Class template
  - Variable templates (since C++14)
  - Variadic templates (since C++11)
  - Template aliases (since C++11)

```
template <typename T>
T max(T &x, T &y)
{
    return x > y ? x : y;
}
```

Template

- **Types of Templates**
  - Function templates
  - Abbreviated function templates (since C++20)
  - Class template
  - Variable templates (since C++14)
  - Variadic templates (since C++11)
  - Template aliases   (since C++11)

```
template<typename T> using Vec = std::vector<T>;
```

```
auto add(auto a, auto b)
{ return a + b; }
```

很像python

```
template<typename T>
constexpr T pi = T(3.14159);
```

```
template <typename... Args>
void print(Args... args)
{
    (std::cout << ... << args);
}
```

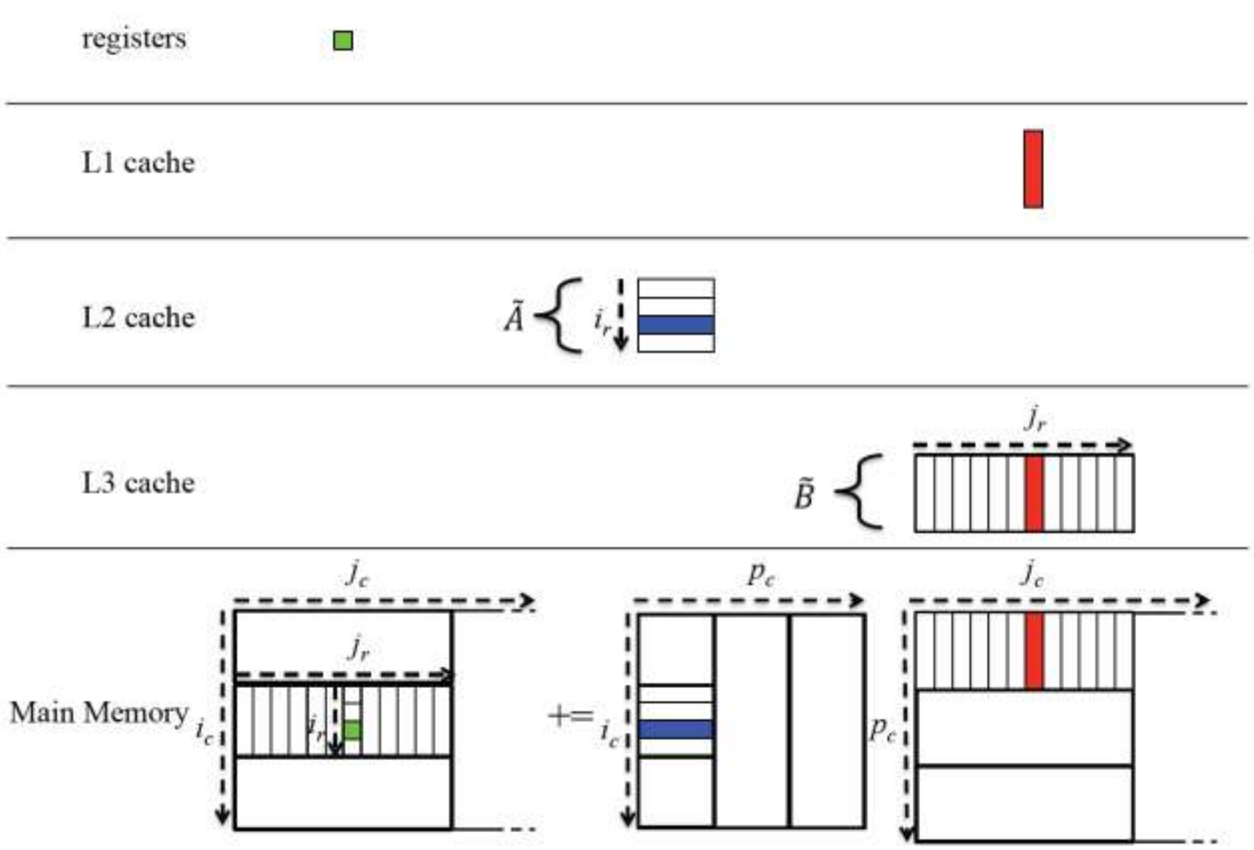
Template

- **Template Parameter**
  - Type Template Parameter `template<typename T>`
  - Non-Type Template Parameter `template<int N>`

Template

- **Template Parameter**
  - Type Template Parameter `template<typename T>`
  - Non-Type Template Parameter `template<int N>` 调参

Template



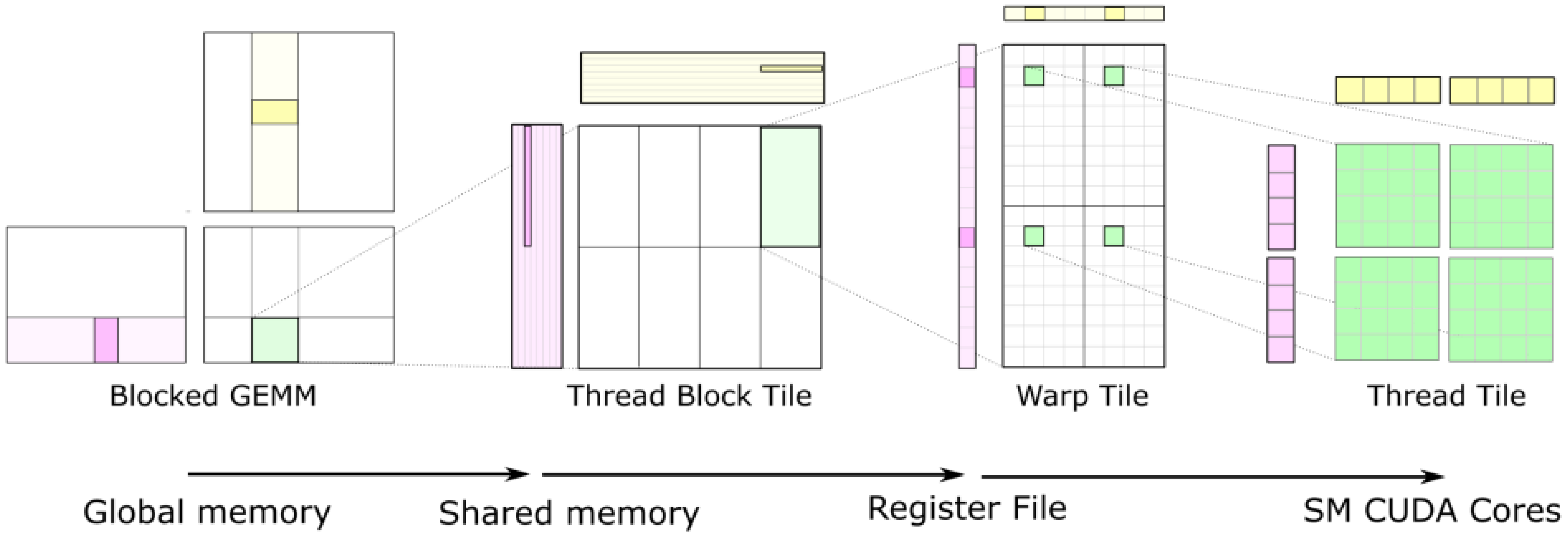
Tuning of GEMM

Template

- **Template Parameter**
  - Type Template Parameter `template<typename T>`
  - Non-Type Template Parameter `template<int N>` 调参
    - CUTLASS (CUDA Templates for Linear Algebra Subroutines)

      https://developer.nvidia.com/blog/cutlass-linear-algebra-cuda/

Template



https://developer.nvidia.com/blog/cutlass-linear-algebra-cuda/

Template

```
/// CUTLASS SGEMM example
__global__ void gemm_kernel(float *C, float const *A, float const *B, int M, int N, int K)
{
    // Define the GEMM tile sizes - discussed in next section
    typedef block_task_policy<128,                          // BlockItemsY: Height in rows of a tile
                              32,                           // BlockItemsX - Width in columns of a tile
                              8,                            // ThreadItemsY - Height in rows of a thread-tile
                              4,                            // ThreadItemsX - Width in columns of a thread-tile
                              8,                            // BlockItemsK - Depth of a tile
                              true,                         // UseDoubleScratchTiles - whether to double-buffer SMEM
                              block_raster_enum::Default    // Block rasterization strategy
                              >
        block_task_policy_t;
```

Template

```
// Define the epilogue functor
typedef gemm::blas_scaled_epilogue<float, float, float> epilogue_op_t;

// Define the block_task type.
typedef block_task<block_task_policy_t,                /// Parameterization of block_task_policy
                   float,                              /// Multiplicand value type (matrices A and B)
                   float,                              /// Accumulator value type (matrix C and scalars)
                   matrix_transform_t::NonTranspose,   /// Layout enumerant for matrix A
                   4,                                  /// Alignment (in bytes) for A operand
                   matrix_transform_t::NonTranspose,   /// Layout enumerant for matrix B
                   4,                                  /// Alignment (in bytes) for B operand
                   epilogue_op_t,                      /// Epilogue functor applied to matrix product
                   4,                                  /// Alignment (in bytes) for C operand
                   true                                /// Whether GEMM supports matrix sizes other than mult
                                                       /// of BlockItems{XY}
                   >
    block_task_t;
```

- Typedef 只是定义了类，没有运行时开销
- 抽象出 block_task_t 需要专业知识

Template

```
    // Declare statically-allocated shared storage
    __shared__ block_task_t::scratch_storage_t smem;

    // Construct and run the task
    block_task_t(reinterpret_cast(&smem),
                 &smem, //
                 A, B,
                 C, //
                 epilogue_op_t(1, 0), M, N,
                 K //
                 )
        .run();
}
```

运行时

Template

- **Template Parameter**
  - Type Template Parameter
  - Non-Type Template Parameter
  - Template Template Parameter
  - Variadic Template Parameters
  - Template Concepts, C++20

```
template<typename T>

template<int N>

template<template<typename> class Container>

template<typename... Args>
void print(Args... args) {
    (std::cout << ... << args) << std::endl;
    // C++17 折叠表达式
}

print(1, 2, 3);
print("hello", "world");
print(2024, "SciComProCoder");
```

Template

- **Template Parameter**
  - Type Template Parameter `template<typename T>`
  - Non-Type Template Parameter `template<int N>`
  - Template Template Parameter `template<template<typename> class Container>`
  - Variadic Template Parameters `template<typename... Args>`
  - Template Concepts, C++20

```
template<typename T>
requires std::integral<T>
```

  - …….

AXPY Revisit

```cpp
template <typename T>
void AXPY(const int n, const T a,
          const T *x, const int incx,
                T *y, const int incy)
{
    for (int i = 0; i < n; ++i)
    {
        y[i * incy] += a * x[i * incx];
    }
}
```

Overloading v.s. Template

- **重载**：穷举所有的可能，编译器自动匹配
- **模板**：编码代码生成的规则，编译器自动生成代码
- **模板匹配**

Overloading v.s. Template

```
void AXPY(const MKL_INT n, const double a,
          const double *x, const MKL_INT incx,
                double *y, const MKL_INT incy);
```

```
template <typename T>
void AXPY(const MKL_INT n, const T a,
          const T *x,       const MKL_INT incx,
                T *y,       const MKL_INT incy);
```

滥用风险：如果 T 是整数也可以编译通过

```
template <typename T1, typename T2>
void AXPY(const MKL_INT n, const T1 a,
          const T2 *x,      const MKL_INT incx,
                T2 *y,      const MKL_INT incy);
```

Overloading v.s. Template

```
void AXPY(const MKL_INT n, const double a,
          const double *x, const MKL_INT incx,
                double *y, const MKL_INT incy);
```

```
template <typename T>
requires (!std::is_same_v<T, double>)
void AXPY(const MKL_INT n, const T a,
          const T *x,      const MKL_INT incx,
                T *y,      const MKL_INT incy);
```

C++ 20

如果 gcc 版本
不够可以注释掉对应代码

```
template <typename T1, typename T2>
requires (!std::is_same_v<T1, T2>)
void AXPY(const MKL_INT n, const T1 a,
          const T2 *x,     const MKL_INT incx,
                T2 *y,     const MKL_INT incy);
```

02-poly-template.cpp

Overloading v.s. Template

```cpp
std::vector<double> x     = {1, 2, 3, 4, 5};
std::vector<double> y     = {6, 7, 8, 9, 10};
double              alpha = 2.0;

AXPY(x.size(), alpha, x.data(), 1, y.data(), 1);
```

有歧义！

```
/home/ningzhangcaltech/Github_Repo/2024SciComProCoder/C++_in_Scientific_Computation/02-poly-specialization-case1.cpp: In function ‘int main()’:
/home/ningzhangcaltech/Github_Repo/2024SciComProCoder/C++_in_Scientific_Computation/02-poly-specialization-case1.cpp:70:13: error: call of overloade
d ‘AXPY(std::vector<double>::size_type, double&, double*, int, double*, int)’ is ambiguous
   70 |         AXPY(x.size(), alpha, x.data(), 1, y.data(), 1);
      |         ~~~~^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
/home/ningzhangcaltech/Github_Repo/2024SciComProCoder/C++_in_Scientific_Computation/02-poly-specialization-case1.cpp:17:6: note: candidate: ‘void Ex
ample1::AXPY(int, T, const T*, int, T*, int) [with T = double]’
   17 | void AXPY(const int n, const T a, const T *x, const int incx, T *y, const int incy)
      |      ^~~~
/home/ningzhangcaltech/Github_Repo/2024SciComProCoder/C++_in_Scientific_Computation/02-poly-specialization-case1.cpp:11:13: note: candidate: ‘void E
xample1::AXPY(size_t, double, const double*, size_t, double*, size_t)’
   11 | inline void AXPY(const MKL_INT n, const double a, const double *x, const MKL_INT incx, double *y, const MKL_INT incy)
```

Overloading v.s. Template

```
void AXPY(const MKL_INT n, const double a,
          const double *x, const MKL_INT incx,
                double *y, const MKL_INT incy);
```

```
template <typename T>
requires (!std::is_same_v<T, double>)
void AXPY(const MKL_INT n, const T a,
          const T *x,      const MKL_INT incx,
                T *y,      const MKL_INT incy);
```

C++ 20 **concepts**

```
template <typename T1, typename T2>
requires (!std::is_same_v<T1, T2>)
void AXPY(const MKL_INT n, const T1 a,
          const T2 *x,     const MKL_INT incx,
                T2 *y,     const MKL_INT incy);
```

02-poly-template.cpp

Specialization

- **Specialization** https://en.wikipedia.org/wiki/Generic_programming#Template_specialization

  A powerful feature of C++'s templates is *template specialization*.

  This allows **alternative implementations** to be provided based on certain characteristics of the parameterized type that is being instantiated.

```
template <>
void AXPY(const MKL_INT n, const double a,
          const double *x, const MKL_INT incx,
                double *y, const MKL_INT incy);

template <typename T>
void AXPY(const MKL_INT n, const T a,
          const T *x,      const MKL_INT incx,
                T *y,      const MKL_INT incy);
```

Specialization

- **Specialization** https://en.wikipedia.org/wiki/Generic_programming#Template_specialization

  A powerful feature of C++'s templates is *template specialization*.

  This allows **alternative implementations** to be provided based on certain characteristics of the parameterized type that is being instantiated.

- **Specialization v.s. Recursion**

  General implementation → Recursion relation
  Alternative implementation → Termination condition

  $$F(n) = F(n-1) + F(n-2)$$

  $$F(0) = 0, F(1) = 1$$

- **Automatic unrolling**

AXPY Revisit

```
template <typename T>
void AXPY(const int n, const T a,
          const T *x, const int incx,
                T *y, const int incy)
{
    for (int i = 0; i < n; ++i)
    {
        y[i * incy] += a * x[i * incx];
    }
}
```

Why Unrolling ?

```cpp
template <typename T>
void AXPY(const int n, const T a,
          const T *x, const int incx,
                T *y, const int incy)
{
    for (int i = 0; i < n; ++i)
    {
        y[i * incy] += a * x[i * incx];
    }
}
```

For 循环是有代价的
可以在编译器知道 n 的大小
且 n 不大的时候，
进行循环展开

Specialization and Auto-Unrolling

```
y[0] += alpha * x[0];
y[1] += alpha * x[1];
y[2] += alpha * x[2];
y[3] += alpha * x[3];
y[4] += alpha * x[4];
```

The compiler must know the length of the vectors

Specialization and Auto-Unrolling

	GFLOPS	Rela.
Plain Cpp	2.098	1.00
Eigen	8.787	4.19
Template	6.458	3.08
ISPC	0.830	0.40
MKL	0.734	0.35

Vec length = 4

Eigen : libeigen3-dev 3.4.0-2ubuntu2
MKL : 2024.0
ISPC : 1.21.1
GCC : 13.1.0 –O2

Specialization and Auto-Unrolling

```
y[0] += alpha * x[0];
y[1] += alpha * x[1];
y[2] += alpha * x[2];
y[3] += alpha * x[3];
y[4] += alpha * x[4];
```

Recursion

```
Func(alpha, x, y):
    *y += alpha * *x
    Func(alpha, x+1, y+1)
```

Termination

```
Func_(alpha, x, y):
    *y = alpha * *x
```

The compiler must know the length of the vectors

Specialization and Auto-Unrolling

```
template <std::floating_point Scalar, int N>
void AXPY(const Scalar a, const Scalar *x, Scalar *y);

template <std::floating_point Scalar>
void AXPY<Scalar, 1>(const Scalar a, const Scalar *x, Scalar *y);
```

函数不能偏特化 但是类可以
类和函数都可以 完全特化

Specialization and Auto-Unrolling

```
template <std::floating_point Scalar, int N>
class automatic_unroll
{
public:
    static void AXPY(const Scalar a, const Scalar *x, Scalar *y)
    {
        *y += a * *x;
        automatic_unroll<Scalar, N - 1>::AXPY(a, x + 1, y + 1);
    }
};

template <std::floating_point Scalar>
class automatic_unroll<Scalar, 1>
{
public:
    static void AXPY(const Scalar a, const Scalar *x, Scalar *y) { *y += a * *x; }
};
```

函数不能偏特化 但是类可以
类和函数都可以 完全特化

Specialization and Auto-Unrolling

```
y[0] += alpha * x[0];  <--  automatic_unroll<double, 5>::AXPY(alpha, x0,   y0  )
y[1] += alpha * x[1];  <--  automatic_unroll<double, 4>::AXPY(alpha, x0+1, y0+1)
y[2] += alpha * x[2];  <--  automatic_unroll<double, 3>::AXPY(alpha, x0+2, y0+2)
y[3] += alpha * x[3];  <--  automatic_unroll<double, 2>::AXPY(alpha, x0+3, y0+3)
y[4] += alpha * x[4];  <--  automatic_unroll<double, 1>::AXPY(alpha, x0+4, y0+4)
```

- Exercises: (1) Fibonacci sequence (2) combination number.

Outline of Polymorphism in C++

1. Overloading, Interface of Linear Algebra Ops
2. Template, Automatic Unrolling
3. **Abstract Base Class, Parallel**

OpenMP/MPI Parallelization

- **大多数算法的并行化：（1）基于任务；（2）并行 for 循环 + 规约操作**
- 并行哪个 for 循环？负载均衡？
- 子任务做什么？（线程）
- 如何规约？
- 执行任务之前线程可能要初始化信息，结束任务之后销毁信息

OpenMP Parallelization

```
inline void OpenMPAlgoBase::run()
{
    /* 初始化 */
    initialize();
    /* 并行计算 */
#pragma omp parallel num_threads(num_of_threads)
    {
        initialize_local();
#pragma omp for schedule(dynamic, 1) nowait
        for (size_t i = begin_indx; i < end_indx; ++i)
        {
            do_task(i);
            /* 合并数据 */
            if (should_upload_data_force()){upload_data_force();}
            else
            { if (should_upload_data()) {upload_data();}}
        }
        finish_local();
    }
    /* 结束 */
    finish();
}
```

OpenMP Parallelization

```cpp
inline void OpenMPAlgoBase::run()
{
    /* 初始化 */
    initialize();
    /* 并行计算 */
#pragma omp parallel num_threads(num_of_threads)
    {
        initialize_local();
#pragma omp for schedule(dynamic, 1) nowait
        for (size_t i = begin_indx; i < end_indx; ++i)
        {
            do_task(i);
            /* 合并数据 */
            if (should_upload_data_force()){upload_data_force();}
            else
            { if (should_upload_data()) {upload_data();}}
        }
        finish_local();
    }
    /* 结束 */
    finish();
}
```

只实现子函数，不用每次重复写
#pragma omp parallel

Algorithm Template

Virtual Function

- **Virtual function**

In object-oriented programming such as is often used in C++ and Object Pascal, a virtual function or virtual method is an inheritable and overridable function or method that is dispatched **dynamically**.

声明，定义，可被子类修改

- **Pure virtual function**

A pure virtual function or pure virutal method is a virtual function that is **required to be implemented** by a derived class if the derived class is not abstract.

只声明，不定义，子类必须定义

Virtual Function

```cpp
class base
{
public:
    virtual std::string name()
    {
        return "base";
    }
    virtual void doSomething() = 0;
};
```

name 方法是一个虚函数，可被子类重写

doSomething 方法是一个纯虚函数，
base 类不可以被实例化，
子类想要被实例化**必须实现**该函数

Virtual Function

```
class derived : public base
{
public:
    virtual std::string name()
    {
        return "derived";
    }
    virtual void doSomething()
    {
        std::cout << "derived do something" <<
std::endl;
    }
};
```

OpenMP Parallelization

```cpp
class OpenMPAlgoBase
{
public:
    virtual void initialize() noexcept = 0;
    virtual void finish() noexcept = 0;
    virtual void initialize_local() noexcept = 0;
    virtual void finish_local() noexcept = 0;
    virtual void do_task(size_t _task_id) noexcept = 0;
    virtual void upload_data() noexcept = 0;
    virtual void upload_data_force() noexcept = 0;
    virtual bool should_update_data() noexcept { return true; }
    virtual bool should_update_data_force() noexcept { return true; }
    /* other code */
    /// ...........
};
```

OpenMP Parallelization

```
inline void OpenMPAlgoBase::run()
{
    /* 初始化 */
    initialize();
    /* 并行计算 */
#pragma omp parallel num_threads(num_of_threads)
    {
        initialize_local();
#pragma omp for schedule(dynamic, 1) nowait
        for (size_t i = begin_indx; i < end_indx; ++i)
        {
            do_task(i);
            /* 合并数据 */
            if (should_upload_data_force()){upload_data_force();}
            else
            { if (should_upload_data()) {upload_data();}}
        }
        finish_local();
    }
    /* 结束 */
    finish();
}
```

OpenMP/MPI Parallelization

Algorithm 13 OpenMP Algorithm Template

```
build_data()                                   ▷ Prepare and verify data
task_queue ← omp_schedule_init()               ▷ Initialize OpenMP scheduling
#pragma omp parallel                           ▷ Begin OpenMP region
  thread_init()
#pragma omp for schedule
for i in task_queue do                         ▷ OpenMP dynamic scheduled loop
    do_task(i)
end for
  thread_finalize()
omp_merge_data()                               ▷ Intra-node reduction
```

Algorithm 14 MPI Algorithm Template

```
build_data()                                   ▷ Same as OpenMP
task_queue ← omp_schedule_init()               ▷ Same as OpenMP
if is_leader_process() then                    ▷ Notify ghost process
    GhostProcessAccess::Send(GhostProcess, MPI_dynamic_task_schedule_signal)
    GhostProcessAccess::Send(GhostProcess, task_queue)
end if
thread_init()
while true do
    next_status ← GhostProcessAccess::request_next_signal()
    if next_status is a sub-queue of task_queue then
        #pragma omp parallel for               ▷ Begin OpenMP region
        for i in next_status do
            do_task(i)
        end for                                ▷ End OpenMP region
    else if next_signal = STOP then            ▷ All chunks done
        break
    else
        throw error "Unknown signal"
    end if
end while
thread_finalize()                              ▷ End OpenMP region
omp_merge_data()                               ▷ Intra-node reduction
mpi_merge_data()                               ▷ Inter-node reduction
```

OpenMP Parallelization

$$\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \cdots = \frac{\pi^2}{6}$$

```
virtual void do_task(size_t _task_id)
{
    auto thread_id = OMP_THREAD_LABEL;
    local_res[thread_id] += 1.0 / (double(_task_id) * double(_task_id));
}

virtual void finish() noexcept
{
    auto res = std::accumulate(local_res.begin(), local_res.end(), 0.0);
    res = sqrt(res * 6.0);
    printf("Pi = %.10f\n", res);
}
```

OpenMP Parallelization

$$\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \cdots = \frac{\pi^2}{6}$$

```cpp
int main()
{
    Pi_Calculator pi_calculator(100000000);
    pi_calculator.Run();
    return 0;
}
```

```
Pi_Calculator::Initialize()
thread 2 finished
thread 7 finished
thread 0 finished
thread 4 finished
thread 6 finished
thread 1 finished
thread 3 finished
thread 5 finished
Pi = 3.1415926442
```

Exercise -- Power Method

- Method for diagonalization, eigenvalue with maximal norm

$$b_{k+1} = \frac{Ab_k}{\|Ab_k\|}$$

```cpp
void run(const std::vector<Scalar>& initialVec)
{
    std::vector<Scalar> vec = initialVec;

    auto norm_init = NORM2_VEC(vec.size(), vec.data(), 1);
    SCAL(vec.size(), 1.0 / norm_init, vec.data(), 1);

    std::vector<Scalar> nextVec(vec.size());
    Scalar              energy = 0, prevEnergy = 0;
```

- (1) finish exercise 1
- (2) finish PowerMethod<Scalar>::run()
- (3) (Optional) CSR MVP
- (4) (Optional) How to unify the implementation of `generateRandomHermitianMatrix` ?

```cpp
for (size_t iter = 0; iter < maxIterations_; ++iter)
{
    mvp_.apply(vec, nextVec);
```

```cpp
auto norm = NORM2_VEC(vec.size(), vec.data(), 1);
SCAL(vec.size(), 1.0 / norm, vec.data(), 1);
```

Power Method

- Method for diagonalization, eigenvalue with maximal norm

$$b_{k+1} = \frac{Ab_k}{\|Ab_k\|}$$

- Matrix Vector Product

```
template <typename Scalar>
class MVP
{
public:
    virtual ~MVP() = default;
    virtual void                apply(const std::vector<Scalar>& vec,
                                          std::vector<Scalar>& result) const = 0;
    virtual std::vector<Scalar> to_fullmat()                               = 0;
};
```

Power Method

- Method for diagonalization, eigenvalue with maximal norm

$$b_{k+1} = \frac{Ab_k}{\|Ab_k\|}$$

- Matrix Vector Product

```
for (size_t iter = 0; iter < maxIterations_; ++iter)
{
    mvp_.apply(vec, nextVec);
```

- Decouple power method and the implementation of MVP !

Challenge!

- Try to implement drivers for matrix diagonalization
- (1) different methods (Davidson, LOBPCG, iVI)
- (2) both exterior root and interior root (low-lying states, core-excitation/ionization)
- (3) for both Hermitian and non-Hermitian/symmetric
- Reference : (a) Davidson method
  (b) LOBPCG
  (c) iVI
  (d) iVI for interior root
  (e) Davidson for non-symmetric matrix
  (f) other

Outline of Advanced Examples

1. Type traits
2. Concept
3. *Expression Templates

Outline of Advanced Examples

1. **Type traits**
2. Concept
3. Expression Templates

Type Traits

- How to communicate with the compiler ?
- How to tell the compiler that an object is a double/float/complex ?
- How to tell the compiler that an object is a matrix/tensor ?
- How to tell the compiler that double/float/complex are floating point types ?
  - **Type traits** define **compile-time** template-based interfaces to query the properties of types.
- In C++ metaprogramming, a **metafunction** receives types and/or integral values, and after performing some logics returns types and/or integral values.

Type Traits

- How to communicate with the compiler ?
- How to tell the compiler that an object is a double/float/complex ? **Internal type**
- How to tell the compiler that an object is a matrix/tensor ?
- How to tell the compiler that double/float/complex are floating point types ?
  - **Type traits** define **compile-time** template-based interfaces to query the properties of types.
- In C++ metaprogramming, a **metafunction** receives types and/or integral values, and after performing some logics returns types and/or integral values.

Type Traits

```
template <typename Scalar>
struct is_floating_point
{
    static const bool value = false;
};

template <>
struct is_floating_point<double>
{
    static const bool value = true;
};

template <>
struct is_floating_point<float>
{
    static const bool value = true;
};
```

默认所有的类都不是浮点数

double / float 标记为浮点数

static const 说明 value 是一个编译时常量

```
is_floating_point<double>::value
is_floating_point_v<double>
```

使用时类似一个函数，以类为输入，可编译期调用

Type Traits + constexpr

```cpp
struct dot_impl
{
    /*!
     * \brief Apply the functor to a and b
     * \param a the left hand side
     * \param b the left hand side
     * \return the dot product of a and b
     */
    template <typename A, typename B>
    static value_t<A> apply(const A& a, const B& b)
    {
        constexpr auto impl = select_dot_impl<A, B>::value;

        if constexpr (impl == dot_impl::BLAS)
        {
            return Backend::blas::dot(a, b);
        }
        else if constexpr (impl == dot_impl::CUBLAS)
        {
            return Backend::cublas::dot(a, b);
        }
        else if constexpr (impl == dot_impl::VEC)
        {
            return Backend::vec::dot(a, b);
        }
        else
        {
            return Backend::standard::dot(a, b);
        }
    }
};
```

- A, B lhs, rhs 的类型
- value_t<T> 返回 T 的标量类型
- select_dot_impl<A,B> 可以根据 A, B

类型种包含的信息在编译器决定调用哪个实现

- If constexpr 编译期就可以判断走哪个分支

Type Traits + constexpr

```cpp
struct dot_impl
{
    /*!
     * \brief Apply the functor to a and b
     * \param a the left hand side
     * \param b the left hand side
     * \return the dot product of a and b
     */
    template <typename A, typename B>
    static value_t<A> apply(const A& a, const B& b)
    {
        constexpr auto impl = select_dot_impl<A, B>::value;

        if constexpr (impl == dot_impl::BLAS)
        {
            return Backend::blas::dot(a, b);
        }
        else if constexpr (impl == dot_impl::CUBLAS)
        {
            return Backend::cublas::dot(a, b);
        }
        else if constexpr (impl == dot_impl::VEC)
        {
            return Backend::vec::dot(a, b);
        }
        else
        {
            return Backend::standard::dot(a, b);
        }
    }
};
```

```cpp
template <typename A, typename B>
constexpr dot_impl select_default_dot_impl()
{
    if (all_dma<A, B> && cblas_enabled)
    {
        return dot_impl::BLAS;
    }

    if (vec_enabled
    && all_vectorizable<vector_mode, A, B>
    && std::same_as<default_intrinsic_type<value_t<A>>,
                    default_intrinsic_type<value_t<B>>>)
    {
        return dot_impl::VEC;
    }

    return dot_impl::STD;
}
```

dma: direct memory access
A, B can be tensor, expression

无运行时代价

Type Traits + constexpr

如果可以，尽可能调用 memcpy/memmove

```
template <typename InputIt, typename OutputIt>
constexpr OutputIt copy(InputIt first, InputIt last, OutputIt d_first)
{
    // c v must be removed first!
    using input_type = std::remove_cv_t<typename std::iterator_traits<InputIt>::value_type>;
    using output_type = std::remove_cv_t<typename std::iterator_traits<OutputIt>::value_type>;

    constexpr bool opt =
        std::is_same_v<input_type, output_type> &&       // judge whether these two types are the same
        std::is_pointer_v<InputIt> &&                    // the input type must be a pointer
        std::is_pointer_v<OutputIt> &&                   // the output type must be a pointer
        std::is_trivially_copy_assignable_v<input_type>; // can be trivially copy assigned!
    // determine when to call memmove !

    return detail::copy_fn<opt>::copy(first, last, d_first);
}
```

opt=true，调用 memcpy/memmove
opt=false, 调用构造函数

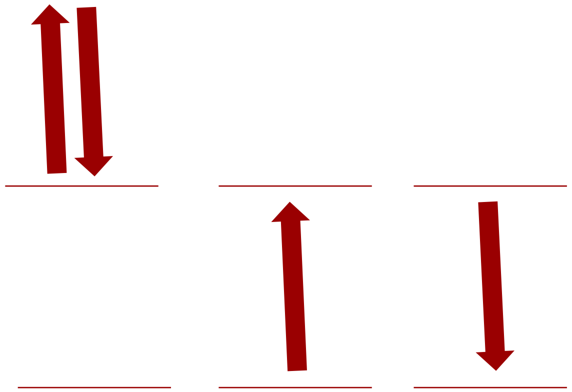
Exercise: learn the implementation of std::copy

Type Traits in MetaWFN package

```
template <>
struct hamiltonian_property<tag::hamiltonian::electron_nonrela>
{
    /* 哈密顿量形式 */
    static constexpr bool has_twobody_term        = true;
    static constexpr bool has_onebody_term        = true;
    static constexpr bool has_oneext_term         = true;
    static constexpr bool has_twoext_term         = true; /// for two excitation case
check whether there is two-body term contributing to the energy
    static constexpr bool has_oneext_twobody_term = true; /// for one excitation case
check whether there is two-body term contributing to the energy
    static constexpr bool has_dg_off_term         = true; /// 指的是同一个 cfg 不同
csf/det 之间有没有矩阵元
    static constexpr bool has_dg_ijji_term        = true; /// 有无 e_{ij,ji} 算符
    /* 守恒量 */
    static constexpr bool spin_conserved = true;
    /* fetch 对角元积分的类型 */
    using fetch_dg_type = tag::hmat_construction::non_relativistic;
};
```

Type Traits in MetaWFN package

```cpp
template <>
struct hamiltonian_property<tag::hamiltonian::Hubbard>
{
    /* 哈密顿量形式 */
    static constexpr bool has_twobody_term        = true;
    static constexpr bool has_onebody_term        = true;
    static constexpr bool has_oneext_term         = true;
    static constexpr bool has_twoext_term         = false;
    static constexpr bool has_oneext_twobody_term = false;
    static constexpr bool has_dg_off_term         = false;
    static constexpr bool has_dg_ijji_term        = false;
    /* 守恒量 */
    static constexpr bool spin_conserved = true;
    /* fetch 对角元积分的类型 */
    using fetch_dg_type = tag::hmat_construction::non_relativistic;
};
```



$$\hat{H} = -t \sum_{i,\sigma} \left(\hat{c}^\dagger_{i,\sigma}\hat{c}_{i+1,\sigma} + \hat{c}^\dagger_{i+1,\sigma}\hat{c}_{i,\sigma}\right) + U\sum_{i} \hat{n}_{i\uparrow}\hat{n}_{i\downarrow},$$

Outline of Advanced Examples

1. Type traits
2. Concept
3. Expression Templates

Concept

- **Concepts and Constraints** https://en.cppreference.com/w/cpp/language/constraints

  - Class templates, function templates, and non-template functions (typically members of class templates) might be associated with a **constraint**, which specifies the requirements on template arguments, which can be used to select the most appropriate function overloads and template specializations.

  - **Named sets of such requirements are called concepts**. Each concept is a predicate, evaluated at compile time, and becomes a part of the interface of a template where it is used as a constraint:

Concept

```
template <typename T>
concept MatrixView = requires(T t) {
    typename T::scalar_type;
    requires std::floating_point<typename T::scalar_type>;
    typename T::iter_type;
    T::constant;
    T::scalar_size;
    {
        t.nrow()
    } -> std::convertible_to<size_t>;
    {
        t.ncol()
    } -> std::convertible_to<size_t>;
    {
        t.data()
    } -> std::convertible_to<typename T::iter_type>;
    {
        t.stride()
    } -> std::convertible_to<size_t>;
};
```

类 T 定义了类型 T::scalar_type

T::scalar_type 是一个浮点数

类 T 定义了static member scarlar_size

t.data() 返回值可以转换成 T::iter_type

RowMajor

Concept

```
template <MatrixView T1, MatrixView T2>
    requires std::same_as<typename T1::scalar_type, typename T2::scalar_type>
void Check_MatrixView(const T1 &t1, const T2 &t2)
{
    std::cout << "Check passed" << std::endl;
}
```

- 要求 T1, T2 满足 MatrixView 要求
- T1::scalar_type, T2::scalar_type 相同

Concept

```cpp
template <std::floating_point T>
class MatrixView_1
{
public:
    // using iter_type = T;
    using scalar_type = T;
    static constexpr bool constant = false;
    static constexpr size_t scalar_size = sizeof(T);

    MatrixView_1(T *data, size_t nrow, size_t ncol, size_t stride = 1) :
data_(data), nrow_(nrow), ncol_(ncol), stride_(stride) {}

    size_t nrow() const { return nrow_; }
    size_t ncol() const { return ncol_; }
    T *data() const { return data_; }
    size_t stride() const { return stride_; }

protected:
    T *data_ = nullptr;
    size_t nrow_ = 0;
    size_t ncol_ = 0;
    size_t stride_ = 0;
};
```

不是 MatrixView，
没有定义 iter_type

Concept

```
template <std::floating_point T>
class MatrixView_2
{
public:
    using iter_type = T;
    using scalar_type = T;
    static constexpr bool constant = false;
    static constexpr size_t scalar_size = sizeof(T);

    MatrixView_2(T *data, size_t nrow, size_t ncol, size_t stride = 1) :
data_(data), nrow_(nrow), ncol_(ncol), stride_(stride) {}

    size_t nrow() const { return nrow_; }
    size_t ncol() const { return ncol_; }
    T *data() const { return data_; }
    size_t stride() const { return stride_; }

protected:
    T *data_ = nullptr;
    size_t nrow_ = 0;
    size_t ncol_ = 0;
    size_t stride_ = 0;
};
```

不是 MatrixView，
定义 iter_type 为 T
但 data() 返回值 T* 转换成不成 T

Concept

```
template <std::floating_point T>
class MatrixView_3
{
public:
    using iter_type = T *;
    using scalar_type = T;
    static constexpr bool constant = false;
    static constexpr size_t scalar_size = sizeof(T);

    MatrixView_3(T *data, size_t nrow, size_t ncol, size_t stride = 1) :
data_(data), nrow_(nrow), ncol_(ncol), stride_(stride) {}

    size_t nrow() const { return nrow_; }
    size_t ncol() const { return ncol_; }
    T *data() const { return data_; }
    size_t stride() const { return stride_; }

protected:
    T *data_ = nullptr;
    size_t nrow_ = 0;
    size_t ncol_ = 0;
    size_t stride_ = 0;
};
```

Concept

```cpp
int main()
{
    // MatrixView_1<int> mat_view_0(nullptr, 10, 10); // template constraint failure
    MatrixView_1<double> mat_view_1(nullptr, 10, 10);
    MatrixView_1<double> mat_view_2(nullptr, 10, 10);
    // Check_MatrixView(mat_view_1, mat_view_2); // note: the required type ‘typename T::iter_type’ is invalid
    //                                           // note: ‘t.data()’ does not satisfy return-type-requirement
    MatrixView_2<double> mat_view_3(nullptr, 10, 10);
    // Check_MatrixView(mat_view_1, mat_view_3);

    MatrixView_3<double> mat_view_4(nullptr, 10, 10);
    MatrixView_3<double> mat_view_5(nullptr, 10, 10);
    Check_MatrixView(mat_view_4, mat_view_5);

    MatrixView_3<float> mat_view_6(nullptr, 10, 10);

    // Check_MatrixView(mat_view_4, mat_view_6); // scalar not the same
}
```

05-concept.cpp

Concept

- The conditions that template parameters need to satisfy can be clearly expressed, making the code more **clear** and **easy to understand**.

- Provides a certain degree of **coding standards** and **enforces** them **at the compiler level**.

- Concepts can check template parameters at compile-time and detect error earlier. If the passed parameters do not meet the requirements of the concept, the compiler will give **clear error messages, which reduces the cost of debugging.**

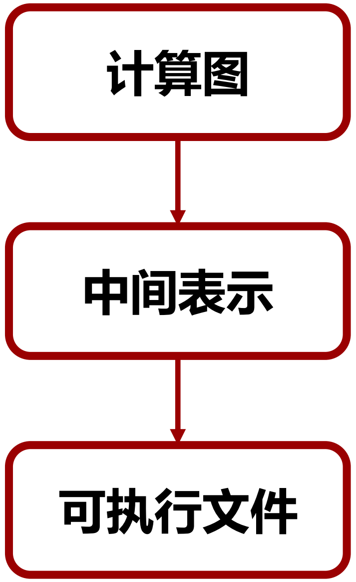
Expression Template

- Build structures at compile time to represent a computation

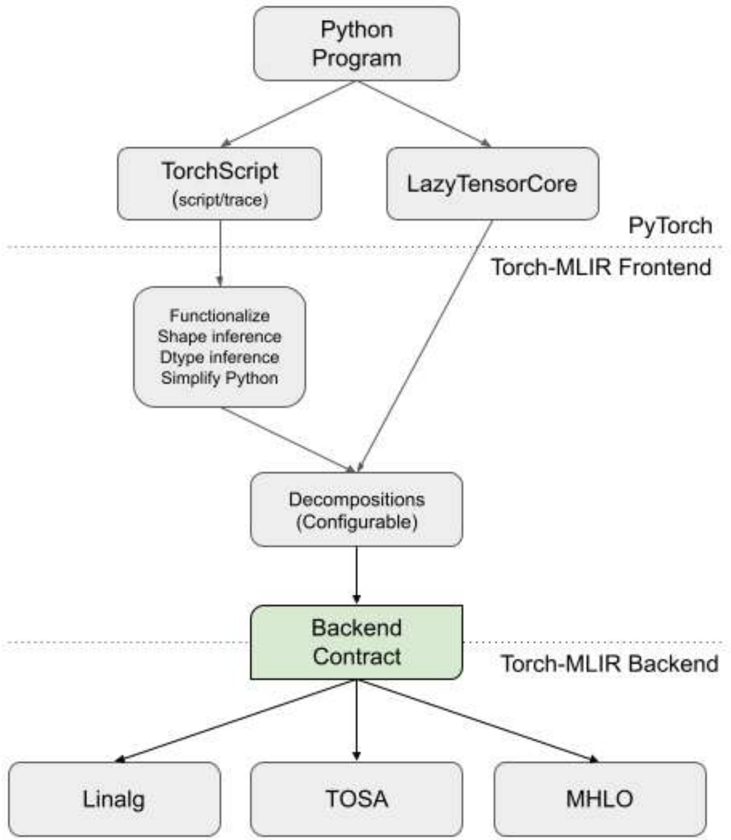
  a+b+c → Sum<Sum<Vec, Vec>,Vec>

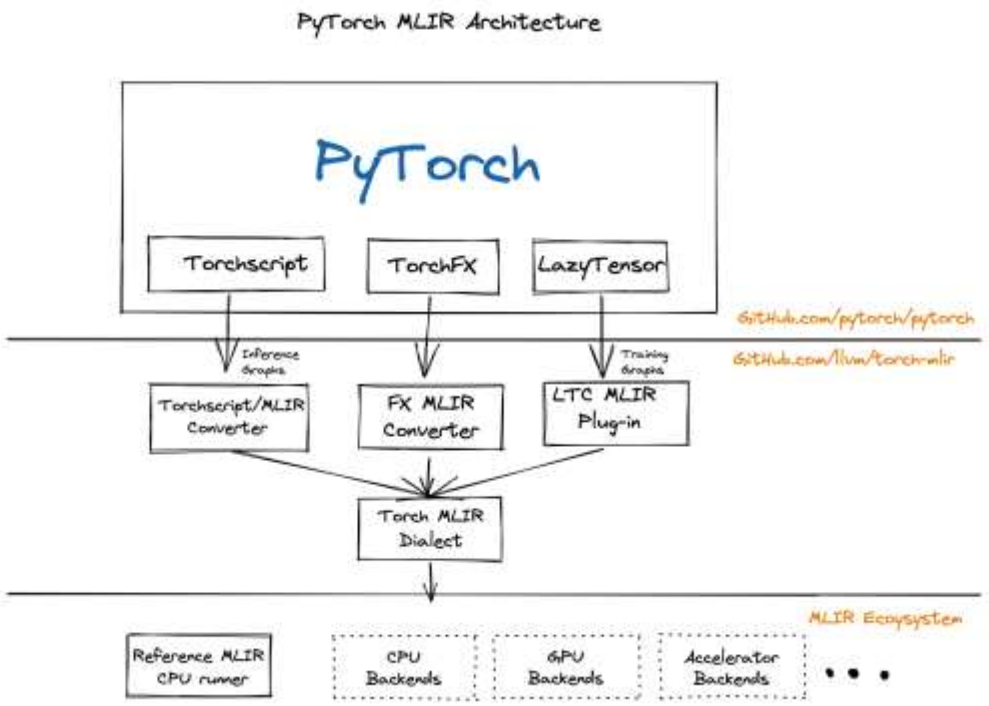
- Lazy evaluation, evaluated only needed

  d = a + b + c



Deep Learning Compiler





Architecture of torch-MLIR

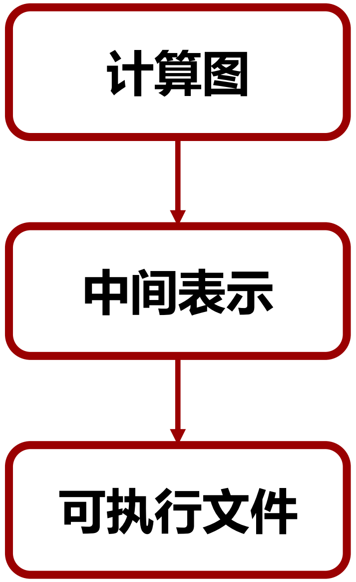
Expression Template

- Build structures at compile time to represent a computation

  a+b+c → Sum<Sum<Vec, Vec>,Vec>

- Lazy evaluation, evaluated only needed

  d = a + b + c



Expression Template

```cpp
// Expression template classes
template <typename LHS, typename RHS>
class BinaryExpression
{
public:
    const LHS& lhs;
    const RHS& rhs;
    BinaryExpression(const LHS& lhs, const RHS& rhs) : lhs(lhs), rhs(rhs) {}
    double operator[](size_t i) const { return lhs[i] + rhs[i]; }
    size_t size() const { return lhs.size(); }
};

template <typename VecType>
class ScalarMult
{
public:
    const VecType& vec;
    double         scalar;
    ScalarMult(const VecType& vec, double scalar) : vec(vec), scalar(scalar) {}
    ScalarMult(double scalar, const VecType& vec) : vec(vec), scalar(scalar) {}
    double operator[](size_t i) const { return vec[i] * scalar; }
    size_t size() const { return vec.size(); }
};
```

- 表示 + 操作，
LHS, RHS 不一定是 vector

(a+b) + c → 't1' + c

[] 操作时进行计算

Expression Template

```cpp
template <typename LHS, typename RHS>
BinaryExpression<LHS, RHS> operator+(const LHS& lhs, const RHS& rhs)
{
    return BinaryExpression<LHS, RHS>(lhs, rhs);
}

template <typename VecType>
ScalarMult<VecType> operator*(double scalar, const VecType& vec)
{
    return ScalarMult<VecType>(vec, scalar);
}

template <typename VecType>
ScalarMult<VecType> operator*(const VecType& vec, double scalar)
{
    return ScalarMult<VecType>(vec, scalar);
}
```

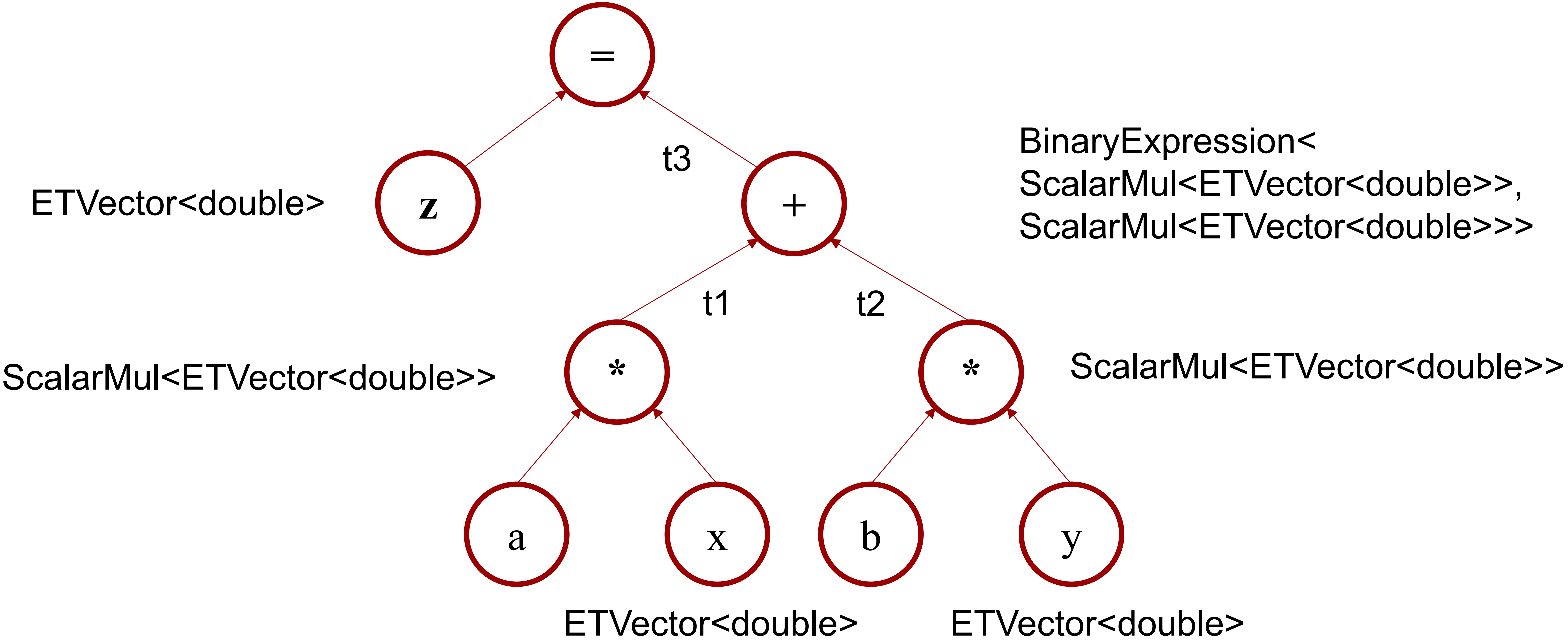
- 重载运算符，但只返回 expression

Expression Template

```cpp
// A Vector wrapper for expression templates
class ETVector
{
public:
    std::vector<double> data;

    ETVector(size_t size) : data(size) {}
    ETVector(const std::vector<double>& vec) : data(vec) {}

    template <typename Expression>
    ETVector& operator=(const Expression& expr)
    {
        for (size_t i = 0; i < data.size(); ++i)
        {
            data[i] = expr[i];
        }
        return *this;
    }

    double operator[](size_t i) const { return data[i]; }
    size_t size() const { return data.size(); }
};
```

- 整体的计算发生在 = 操作

Expression Template



Expression Template

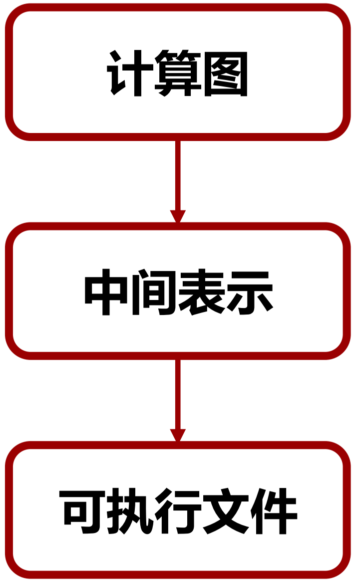
- Build structures at compile time to represent a computation

  a+b+c → Sum<Sum<Vec, Vec>,Vec>

- Lazy evaluation, evaluated only needed

  d = a + b + c

- See all the operands and target
- Automatically composite expressions for element-wise operations
- Optimization → template transformation
- **Hard for compiler to optimize (vectorization)**



Expression Template

$$z_i = a \times \frac{x_i + y_i}{x_i} + y_i + \frac{x_i}{b}$$

```
for (size_t i = 0; i < n; ++i)
    {
        z[i] = a * (x[i] + y[i]) / y[i] + y[i] + x[i] / b;
    }
```

```
double b_inv = 1.0 / b;
for (size_t i = 0; i < n; ++i)
    {
        z[i] = a * (x[i] + y[i]) / y[i] + y[i] + x[i] * b_inv;
    }
```

```
z1 = a * (x1 + y1) / y1 + y1 + x1 / b;
```

Expression Template

$$z_i = a \times \frac{x_i + y_i}{x_i} + y_i + \frac{x_i}{b}$$

N = 10,000,000 -O3, gcc 13.1.0 微秒

	C (version 1)	C (version 2)	E.T.
no vec	14218	8898	15030
avx	8165	7896	5980
avx2	7983	7775	5690
avx512	7872	7612	5362
sse3	14691	8932	7953

Expression Template

$$z_i = a \times \log(\frac{x_i + y_i}{x_i}) + \exp(y_i + \frac{x_i}{b})$$

```
for (size_t i = 0; i < n; ++i)
    {
        z[i] = a * log((x[i] + y[i]) / y[i]) + exp(y[i] + x[i] / b);
    }
```

```
z1 = a * log((x1 + y1) / y1) + exp(y1 + x1 / b);
```

Expression Template

$$z_i = a \times \log(\frac{x_i + y_i}{x_i}) + \exp(y_i + \frac{x_i}{b})$$

N = 10,000,000 -O3, gcc 13.1.0

微秒

	C	E.T.	
no vec	109676	113283	1.033
avx	109814	109944	1.001
avx2	109864	110282	1.004
avx512	108246	115419	1.066
sse3	110564	111415	1.008

Summary

- C++ is extremely powerful in developing high-performance generic scientific computation package

- C++ is all you need !

- Polymorphism in C++ : (1) overloading (2) template (3) virtual function

- More advanced features/idioms in (modern) C++

  (1) type traits; (2) concept; (3) expression templates

  (3) CRTP ; (4) Mixin; (6) SFINAE …….

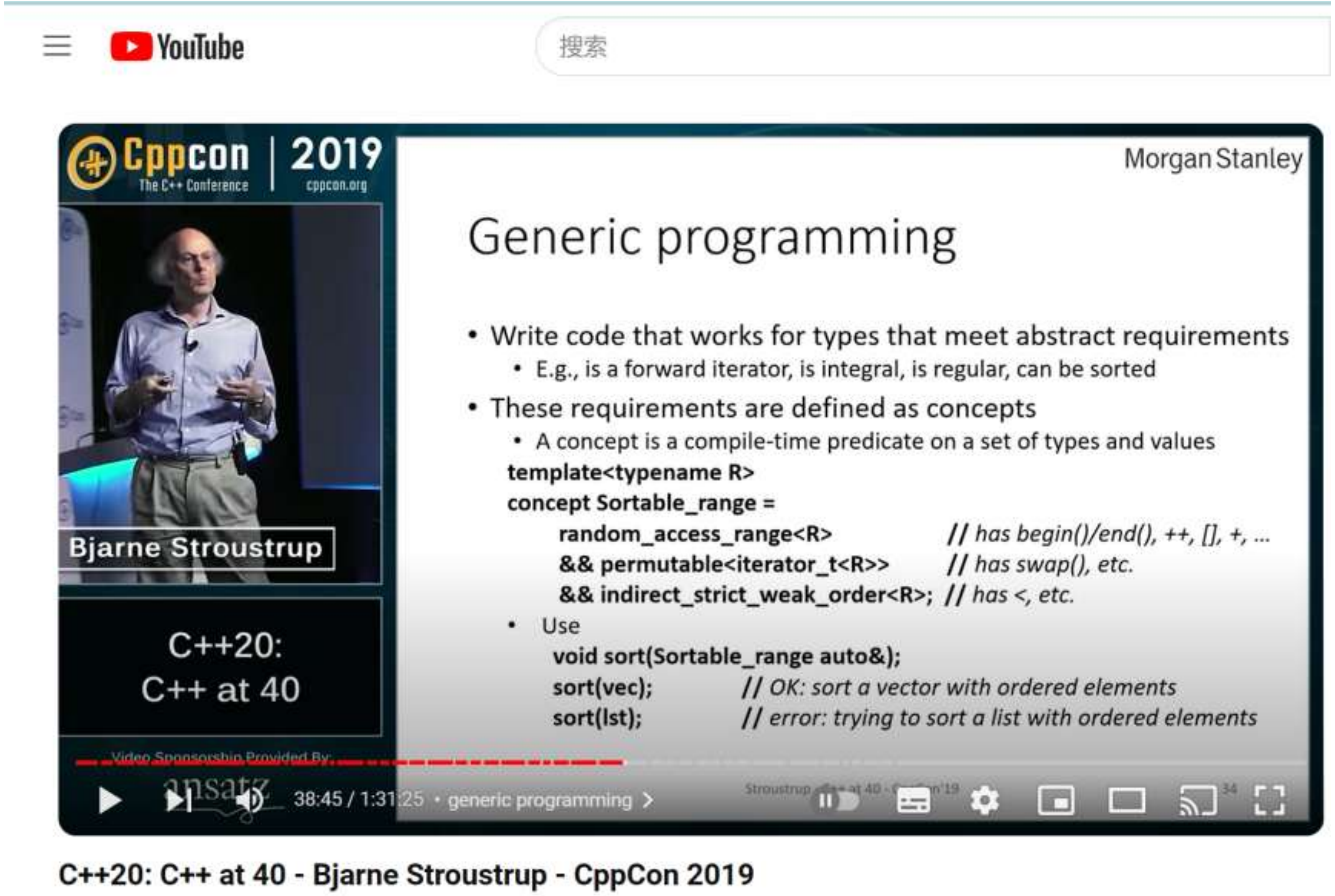
Further Reference

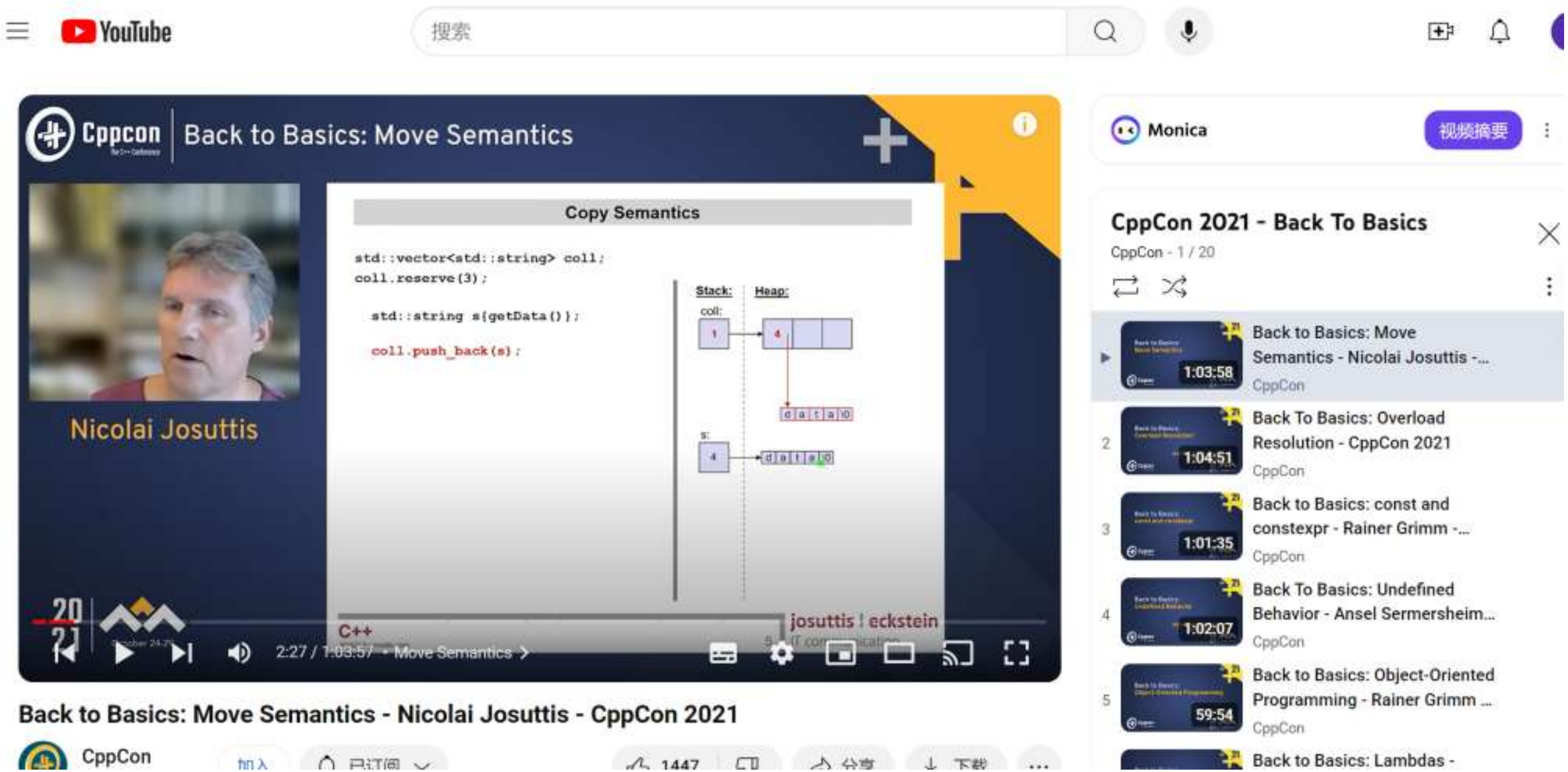
- **YouTube Channel**

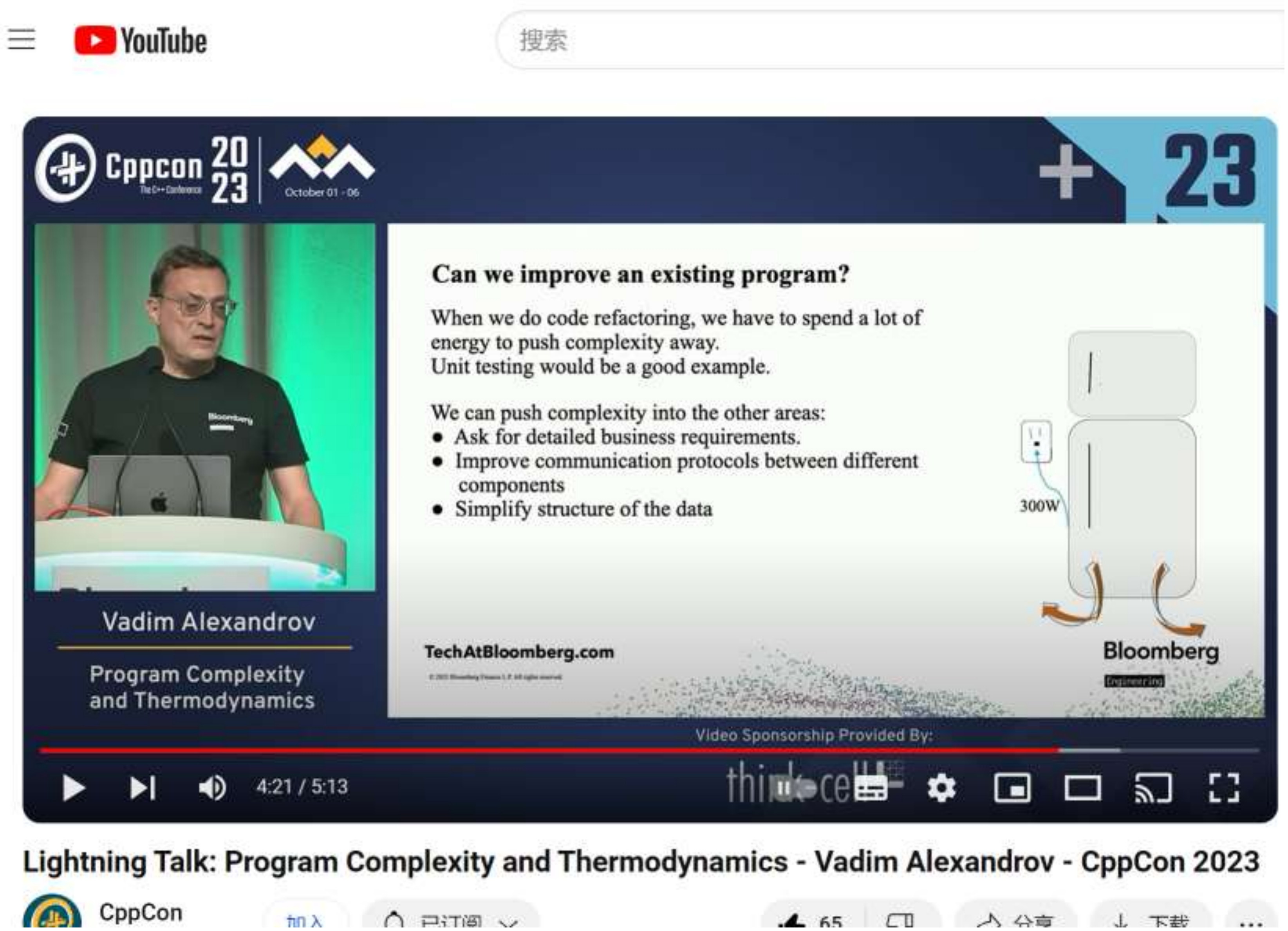
  https://www.youtube.com/@CppCon

  https://cppcon.org/

  https://www.youtube.com/@cppweekly


YouTube
搜索
CppCon The C++ Conference | 2019 cppcon.org
Morgan Stanley
Generic programming
• Write code that works for types that meet abstract requirements
• E.g., is a forward iterator, is integral, is regular, can be sorted
• These requirements are defined as concepts
• A concept is a compile-time predicate on a set of types and values
template<typename R>
concept Sortable_range =
random_access_range<R> // has begin()/end(), ++, [], +, …
&& permutable<iterator_t<R>> // has swap(), etc.
&& indirect_strict_weak_order<R>; // has <, etc.
• Use
void sort(Sortable_range auto&);
sort(vec); // OK: sort a vector with ordered elements
sort(lst); // error: trying to sort a list with ordered elements
Bjarne Stroustrup
C++20:
C++ at 40
38:45 / 1:31:25 • generic programming
C++20: C++ at 40 - Bjarne Stroustrup - CppCon 2019


YouTube
搜索
Cppcon
Back to Basics: Move Semantics
Nicolai Josuttis
Copy Semantics
std::vector<std::string> coll;
coll.reserve(3);
std::string s{getData()};
coll.push_back(s);
Stack:
Heap:
coll:
s:
josuttis | eckstein
C++
2:27 / 1:03:57 • Move Semantics
Back to Basics: Move Semantics - Nicolai Josuttis - CppCon 2021
CppCon
1447
Monica
视频摘要
CppCon 2021 - Back To Basics
CppCon - 1 / 20
Back to Basics: Move Semantics - Nicolai Josuttis -...
CppCon
1:03:58
Back To Basics: Overload Resolution - CppCon 2021
CppCon
1:04:51
Back to Basics: const and constexpr - Rainer Grimm -...
CppCon
1:01:35
Back To Basics: Undefined Behavior - Ansel Sermersheim...
CppCon
1:02:07
Back to Basics: Object-Oriented Programming - Rainer Grimm ...
CppCon
59:54
Back to Basics: Lambdas -



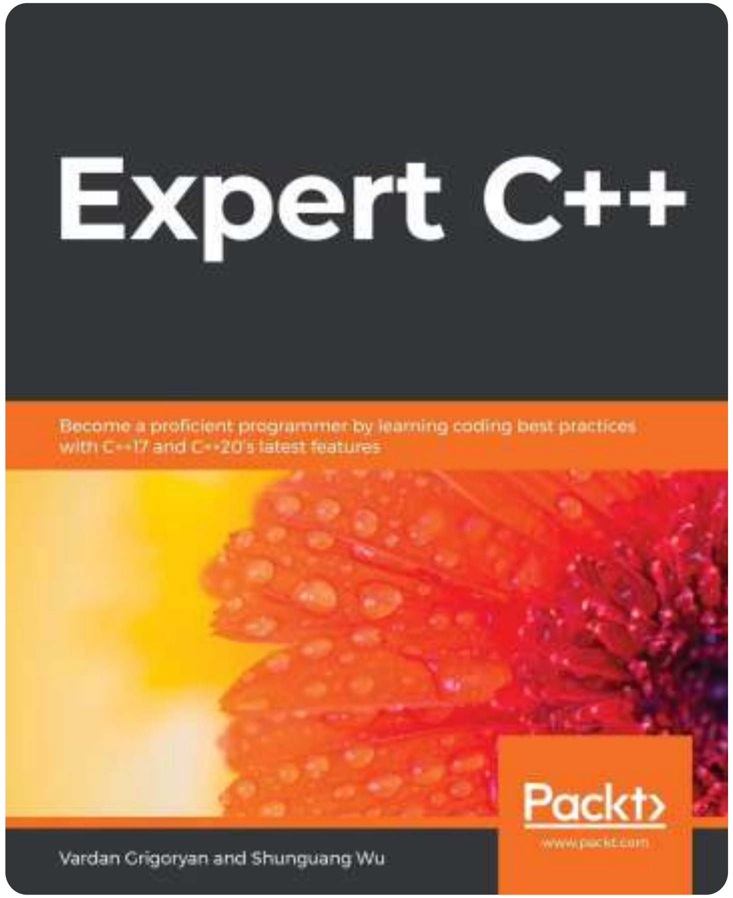
搜索
Cppcon 2023
October 01 - 06
23
Vadim Alexandrov
Program Complexity and Thermodynamics
Can we improve an existing program?
When we do code refactoring, we have to spend a lot of energy to push complexity away.
Unit testing would be a good example.
We can push complexity into the other areas:
• Ask for detailed business requirements.
• Improve communication protocols between different components
• Simplify structure of the data
300W
TechAtBloomberg.com
Bloomberg
Video Sponsorship Provided By:
4:21 / 5:13
Lightning Talk: Program Complexity and Thermodynamics - Vadim Alexandrov - CppCon 2023
CppCon
65

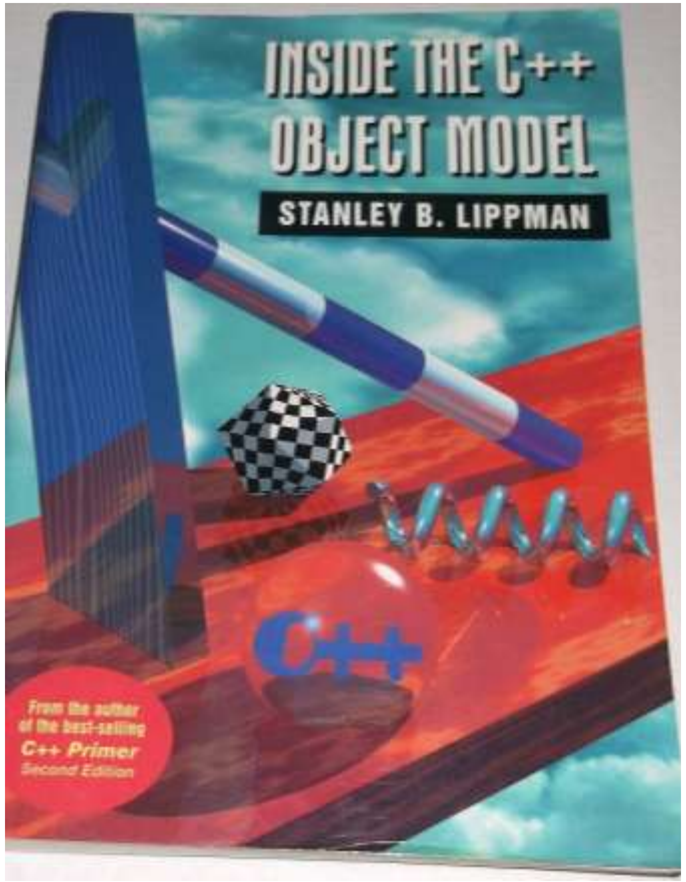
Further Reference

- **Github Repo**

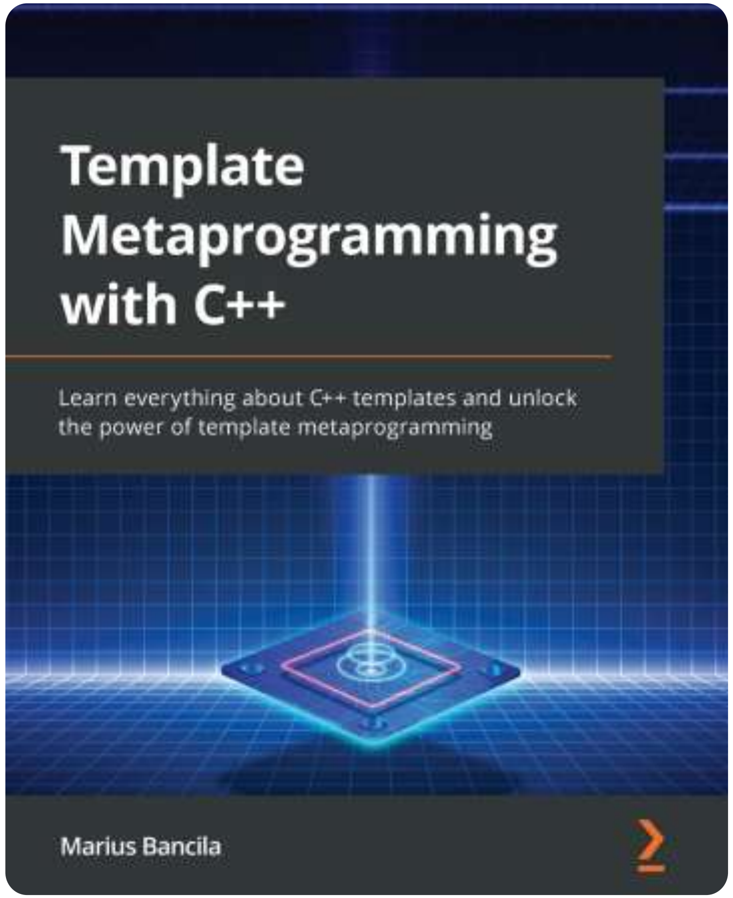
https://github.com/BenBrock/matrix-cpos	稀疏矩阵的现代 C++ 实现
https://github.com/kokkos	CPU/GPU 的统一实现
https://github.com/dpilger26/NumCpp	Numpy 的 C++ 版本
https://github.com/p12tic/libsimdpp	抽象 SIMD 指令
https://github.com/wichtounet/etl	现代高性能表达式模板
https://github.com/boostorg/hana	现代模板元编程库
https://github.com/dmlc/mshadow	容易理解的高性能表达式模板

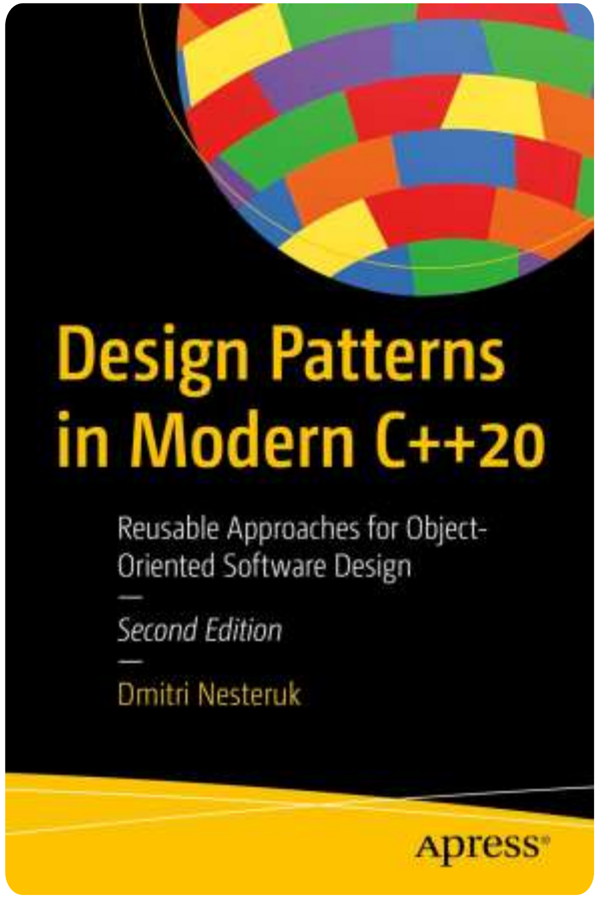
Further Reference



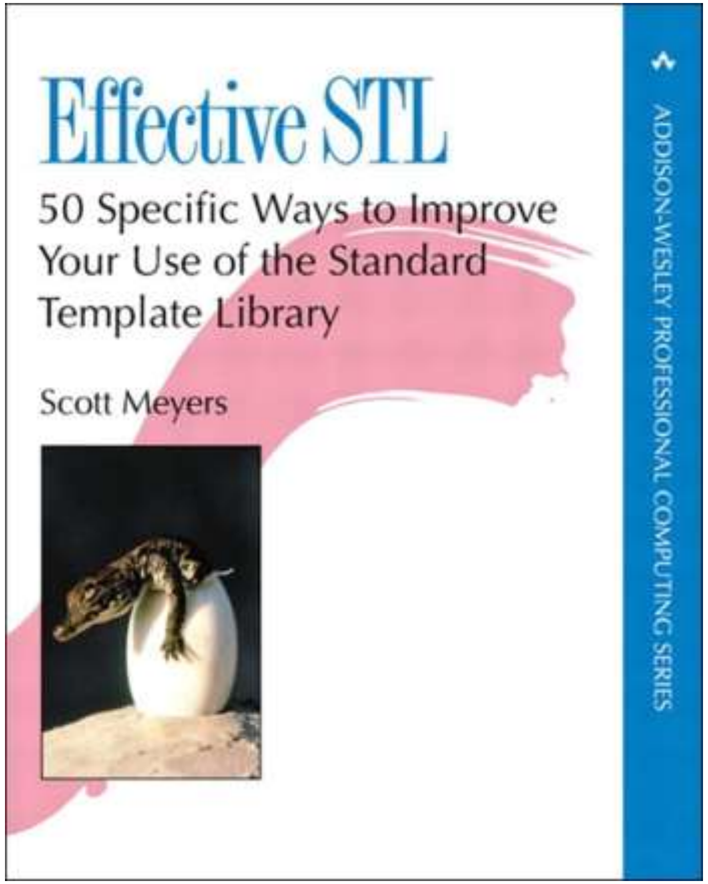


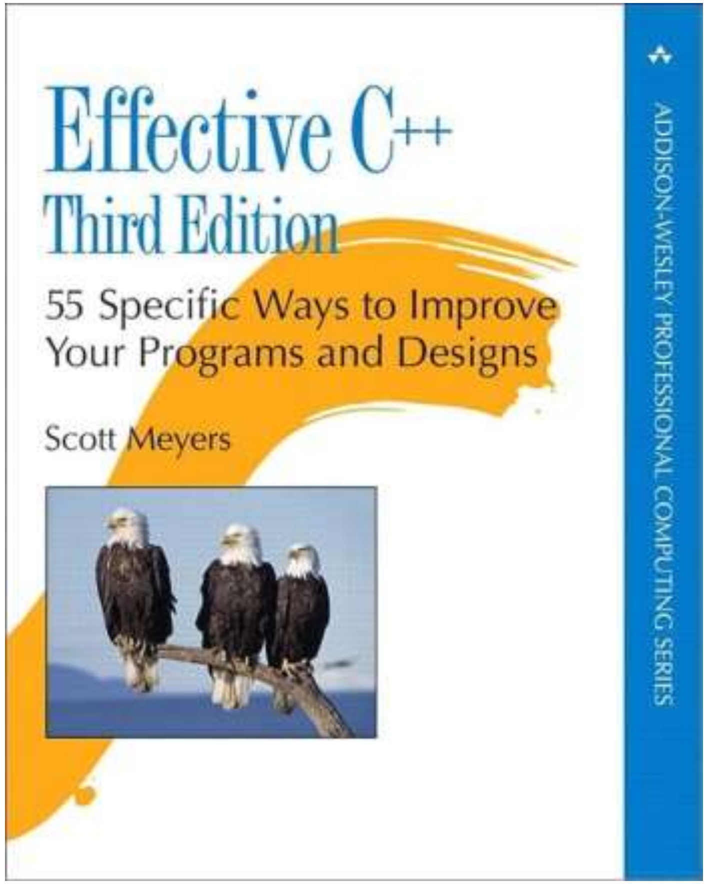
Further Reference

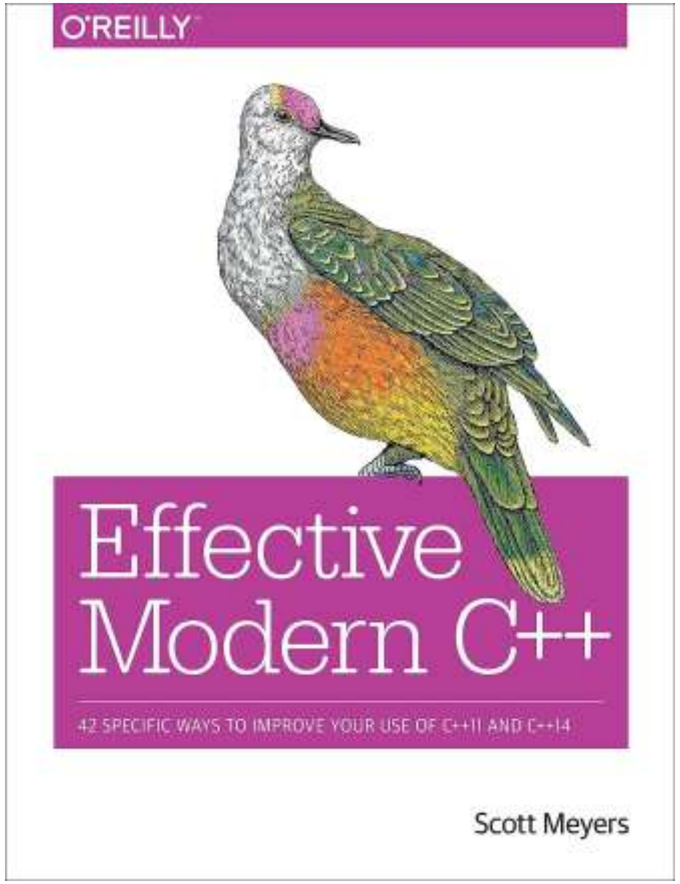




Further Reference







Summary

- C++ is extremely powerful in developing high-performance generic scientific computation package

- C++ is all you need !

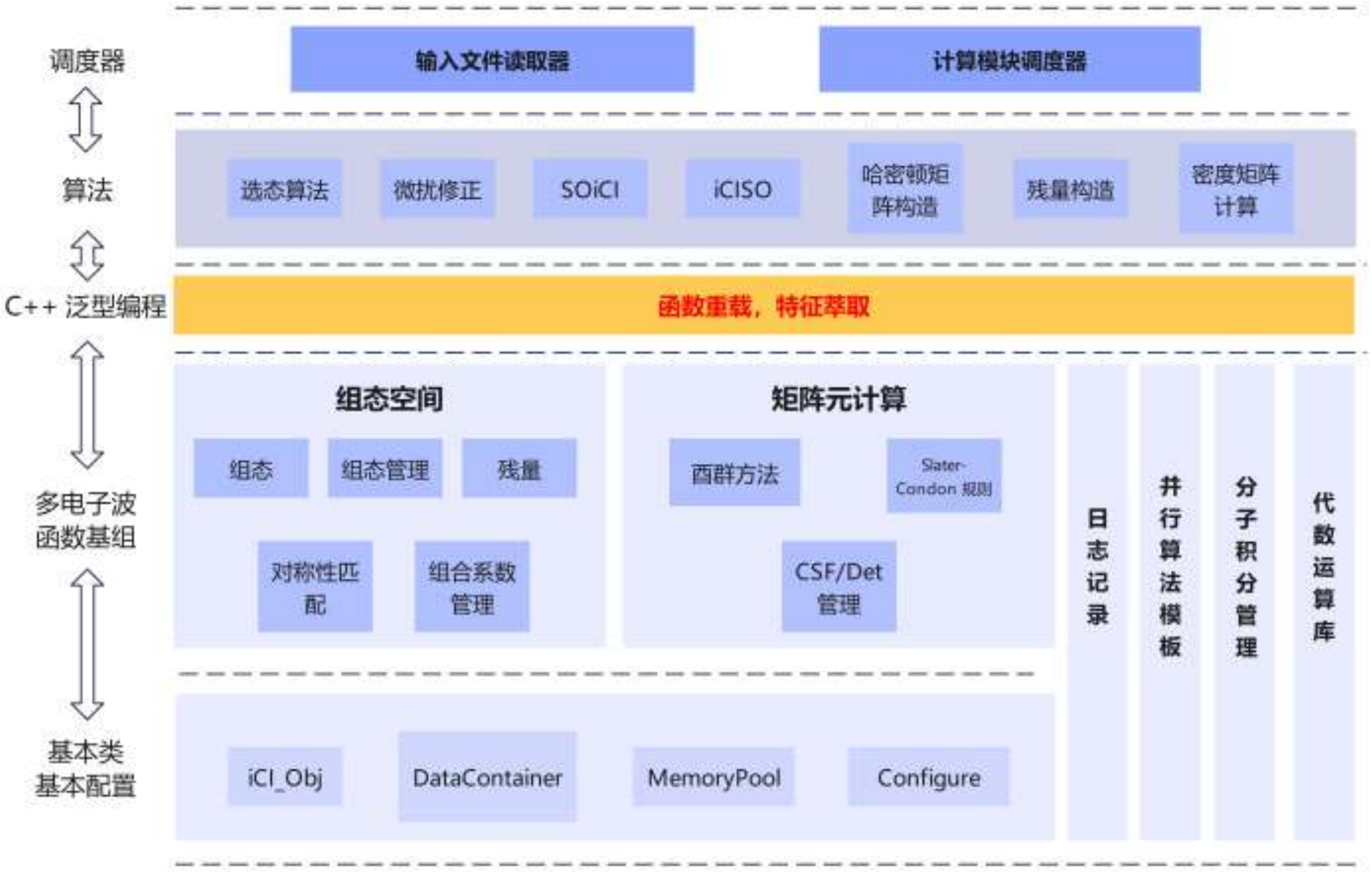
- Polymorphism in C++ : (1) overloading (2) template (3) virtual function

- More advanced features/idioms in (modern) C++

  (1) type traits; (2) concept; (3) expression templates

  (3) CRTP ; (4) Mixin; (6) SFINAE …….

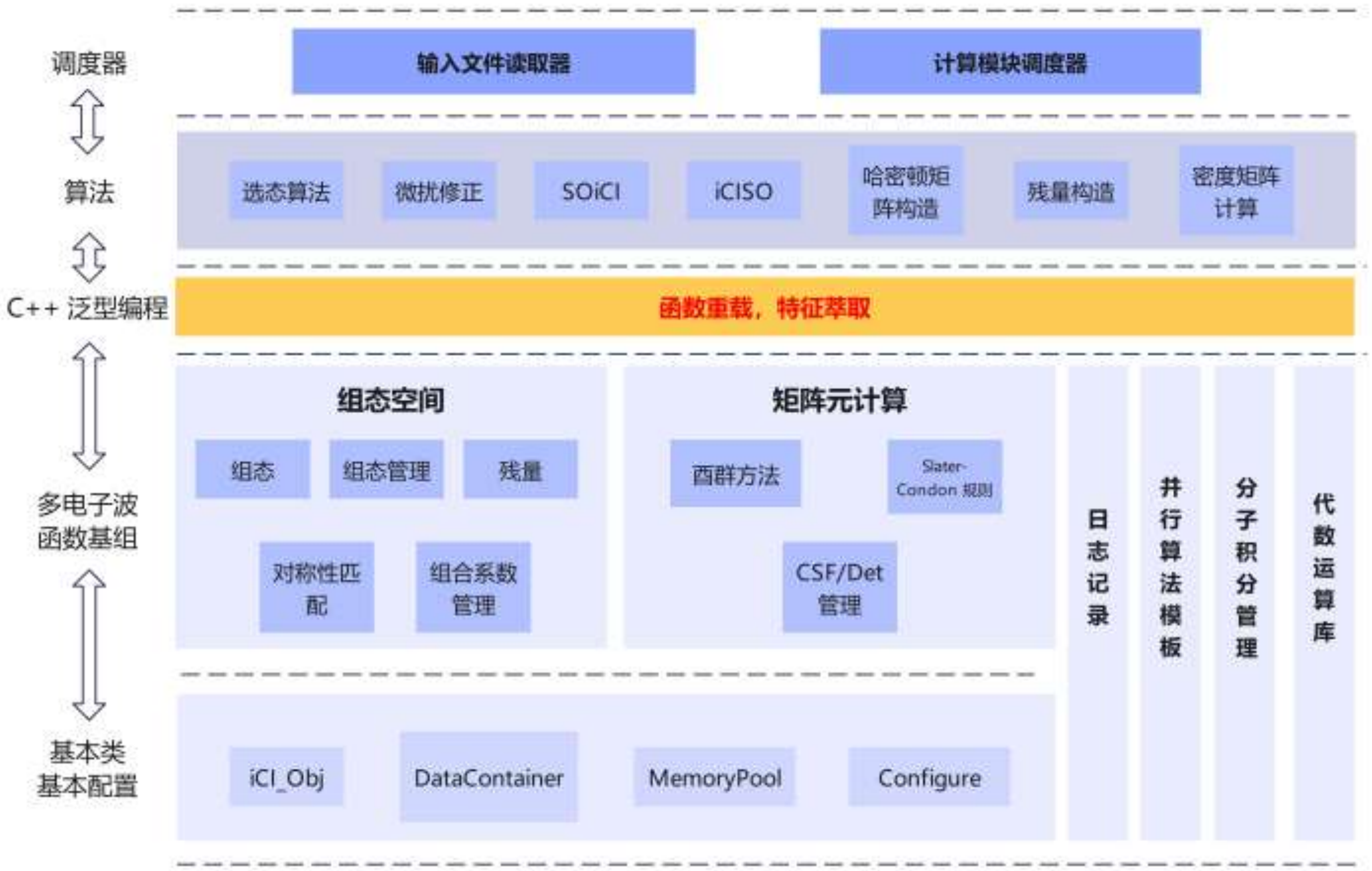
MetaWFN Package



MetaWFN Package

Quantum Chemistry

(Hamiltonian, molecular symmetry)



Algorithm/Method

(HF, CI, CC, PT, CASSCF, QM/MM)

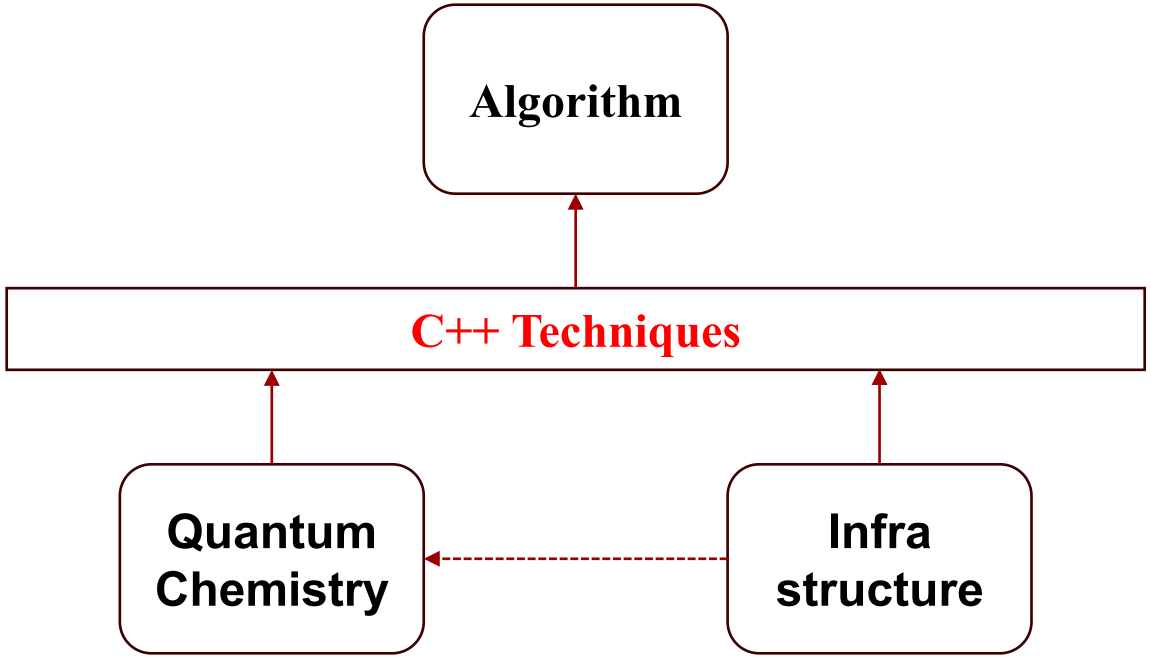
Infra Structure

(linear algebra, parallelization)

MetaWave Package

- Functionality of **MetaWave**
  - iCIPT2, iCISO, SOiCI, 4C-iCIPT2
  - Density matrices
  - Natural orbital construction
  - Core-valence separation Approximation
  - Time evolution
  - Parallelization : OpenMP, MPI, (GPU)
  - ………….

Complexity of Quantum Chemistry Package



Skeleton of MetaWFN

Automatic Compilation of Quantum Chemistry Methods