基于太湖之光开发格点 QCD 计算程序 ——材料的整理与问题的提出

宫明

中科院高能所
高性能计算专项项目组

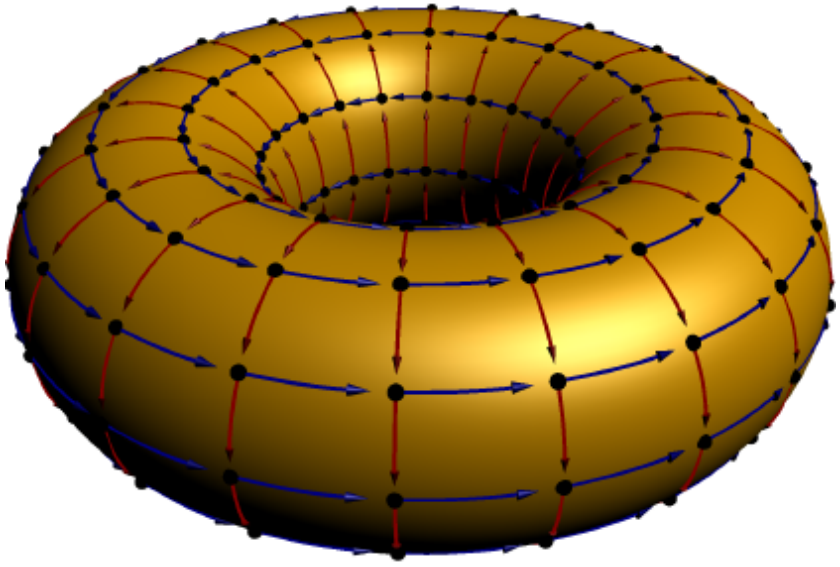
2017.12.12

提纲

提纲

放在格子上的基本自由度

- 我们把一块四维时空切成四维的格点阵列，并把每个维度的两端分别粘起来。(右图为二维示意)
- 在每个格点上放置表示夸克的费米子场量，它是含有 3 个色分量和 4 个旋量分量的格拉斯曼数。我们在计算机上具体操作的是12 个复数构成的向量。
- 在相邻点之间的连接上放置表示胶子的规范场量，它可以写成色空间的3×3 复数矩阵。每个格点有 8 个这样的矩阵，连接到 8 个邻居。



$$\phi(x,y,z,t) = \begin{pmatrix} \begin{pmatrix} d_{11} \\ d_{12} \\ d_{13} \end{pmatrix} \\ \begin{pmatrix} d_{21} \\ d_{22} \\ d_{23} \end{pmatrix} \\ \begin{pmatrix} d_{31} \\ d_{32} \\ d_{33} \end{pmatrix} \\ \begin{pmatrix} d_{41} \\ d_{42} \\ d_{43} \end{pmatrix} \end{pmatrix}, \quad U_\mu(x,y,z,t) = \begin{pmatrix} c_{11} c_{12} c_{13} \\ c_{21} c_{22} c_{23} \\ c_{31} c_{32} c_{33} \end{pmatrix}$$

- 对这个甜甜圈世界的一个“状态”的描述就是给每个点赋予费米子场量、每个连接上赋予规范场量。
- 不同的状态实际出现的概率不同，这个概率正比于（不严格类比）：

$$e^{-\beta S_g(U)+\bar{\Psi}M(U)\Psi}$$

其中 S_g 是所有规范场量 $U_\mu(\vec{x})$ 的一个泛函，M 是与 $U_\mu(\vec{x})$ 有关的一个矩阵，这两者定义了格点上的物理模型。

- 把所有的状态积分起来得到配分函数，格拉斯曼数 $\bar{\Psi}$ 和 Ψ 可以积掉，剩下 $U_\mu(\vec{x})$ 用来表示规范场状态，称为“组态”。

$$Z = \int dU\ d\bar{\Psi}\ d\Psi\ e^{-\beta S_g(U)+\bar{\Psi}M(U)\Psi} = \int dU\ e^{-\beta S_g(U)} \det M(U)$$

- 则任何物理量 $O(U, G_q(U))$ 的期望值可以写成：

$$\langle O\rangle = \frac{1}{Z}\int dU\ O(U, G_q(U)) e^{-\beta S_g(U)} \det M(U)$$

其中 $G_q(U) = M^{-1}(U)$ 表示夸克传播子。

用蒙特卡洛算法计算这个积分

只需要两个步骤

- 随机生成规范场组态 $\{U_\mu(\vec{x})\}$ 使得出现概率：

$$P(U) \propto e^{-\beta S_g(U)} \det M(U)$$

- 计算可观测量 O

$$\langle O \rangle = \frac{1}{N} \sum_{\{U\}} O(U, G_q(U))$$

- 问题的关键：$\det M(U)$和$M^{-1}(U)$是难以被直接计算的
- 复矩阵 M 的行、列数均为 $L_x \times L_y \times L_z \times L_t \times 4 \times 3 \approx O(10^5 - 10^8)$
- M 的形式有若干种。Wilson 费米子以及其改良版只包含近邻相互作用，矩阵是稀疏的。overlap 费米子是非稀疏的。

问题的转化

- overlap 的 M_{ov} 可以用 Wilson 的 M_w 的分式级数展开来逼近，问题可以转化为类似 $M_w^{-1}(U)$ 的稀疏矩阵求逆计算。
- 我们采用所谓“随机伪费米场”的办法来估计 $\det M(U)$，问题也可以转化为求解 $M^{-1}(U)$。
- 对于 $M^{-1}(U)$，我们每次只计算它的若干列而非整个逆矩阵也就够了。问题转化为求解线性方程组：

$$M\,x \;=\; b$$

- 求解线性方程组只需要在 $\{b, Mb, M^2 b, M^3 b, \ldots\}$ 张开的 Krylov 子空间里迭代逼近即可达到满意的精度。
- 在共轭梯度法等 Krylov 子空间算法中，关键的操作为“矩阵乘以向量”和“两个向量内积”。前者是主要的计算热点，后者涉及全局归约操作。
- 于是我们首先要高效实现“矩阵乘以向量” $M\,v$ 的操作。

- Wilson 费米子矩阵的形式：

$$M_{w,\vec{x}\vec{y}} = \frac{1}{2\kappa}\delta_{\vec{x}\vec{y}}1^{(12)} - \frac{1}{2}\sum_{\mu}\left[(1^{(4)}-\gamma_\mu)U_\mu(\vec{x})\delta_{\vec{x}+\mu,\vec{y}} + (1^{(4)}+\gamma_\mu)U_\mu^\dagger(\vec{x}-\mu)\delta_{\vec{x}-\mu,\vec{y}}\right]$$

  其中 μ 表示四个时空方向；$\vec{x}$ 和 $\vec{y}$ 是两个 4-矢量时空坐标；
  $U_\mu(\vec{x})$ 表示从 x 点向 μ 方向的规范场连接，它是 3×3 的复矩阵；$U^\dagger$ 是它的转置复共轭；
  γ_μ 是 4×4 的稀疏复矩阵（见下页）。

- （我们计划采用 clover 改进的 Wilson 费米子作用量，多一个在对角元上的系数。另外，我们采用非对称格距，会在后面两项上加一些系数。对于算法实现来说，这些细节并不重要。）
- 矩阵的对角元部分的计算很简单，一般可以单独处理。
- 非对角元上的计算是关键，定义：

$$\not{\emptyset} = \sum_{\mu}\left[(1^{(4)}-\gamma_\mu)U_\mu(\vec{x})\delta_{\vec{x}+\mu,\vec{y}} + (1^{(4)}+\gamma_\mu)U_\mu^\dagger(\vec{x}-\mu)\delta_{\vec{x}-\mu,\vec{y}}\right]$$

QDP++/MILC/CPS 使用的 DeGrandRossi 基：

$$\gamma_0 = \begin{pmatrix} 0 & 0 & 0 & i \\ 0 & 0 & i & 0 \\ 0 & -i & 0 & 0 \\ -i & 0 & 0 & 0 \end{pmatrix} \gamma_1 = \begin{pmatrix} 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \end{pmatrix}$$

$$\gamma_2 = \begin{pmatrix} 0 & 0 & i & 0 \\ 0 & 0 & 0 & -i \\ -i & 0 & 0 & 0 \\ 0 & i & 0 & 0 \end{pmatrix} \gamma_3 = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

也可以换用其他基，使某一方向的矩阵为对角的 1 或-1，则这个方向的 $1 \pm \gamma_\mu$ 少 4 个非零元。或者在计算中组合凑对也可以达到类似效果。

最终，首要的问题转化成：

如何在太湖之光上高效的并行计算 $\not{D}\, v$

代码参考

http://ihepbox.ihep.ac.cn/ihepbox/index.php/s/bnpq1OUgPQfj52M

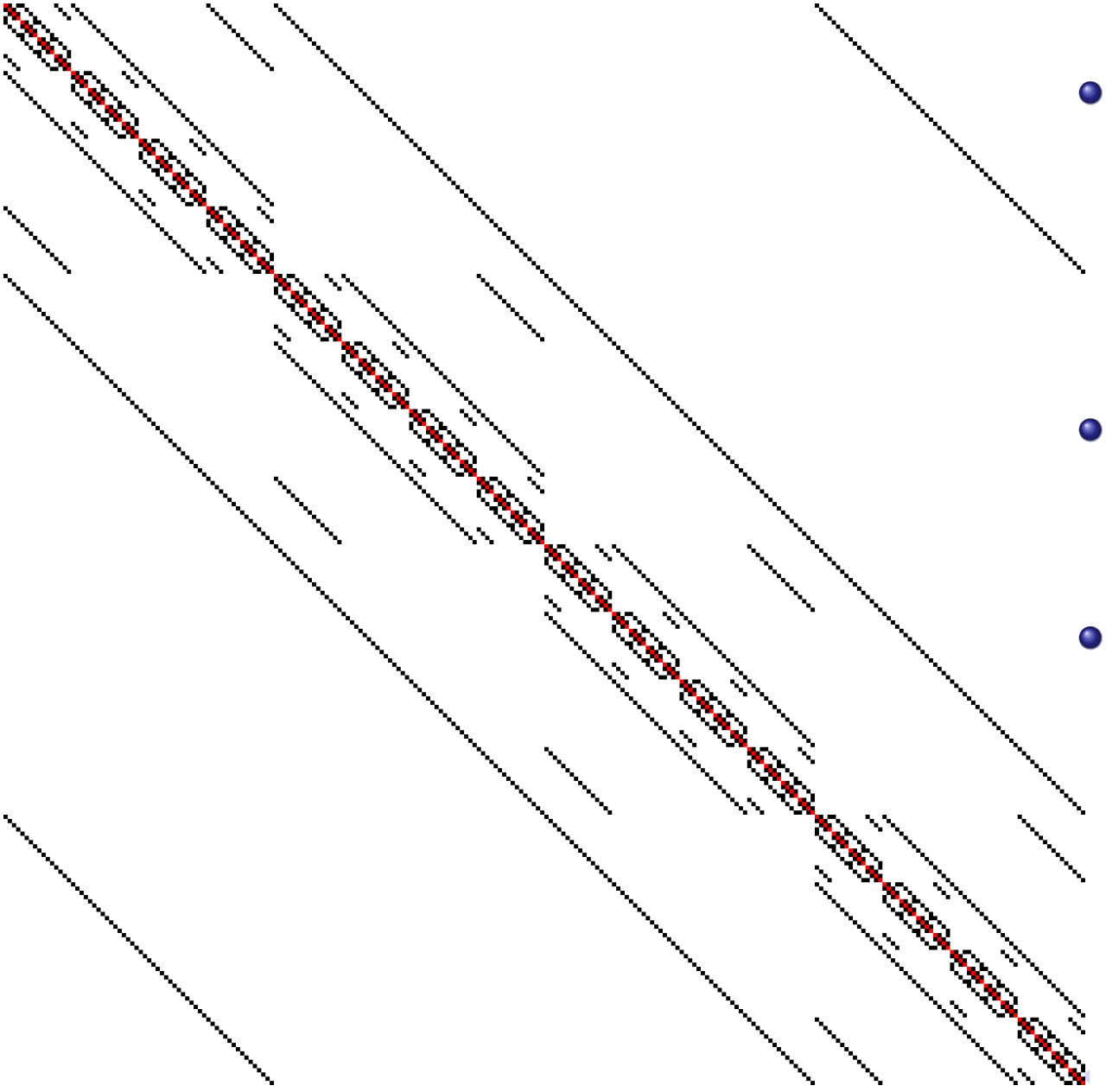
```
typedef struct
{
  Float re, im;
}complex;

typedef struct
{
  complex c1, c2, c3;
}su3_vector;

typedef struct
{
  complex c11, c12, c13;
  complex c21, c22, c23;
  complex c31, c32, c33;
}su3;

typedef struct
{
  su3_vector c1,c2,c3,c4;
}spinor;

// gauge and spinor field
typedef su3 su3_field[NT][NZ][NY][NX][4];
typedef spinor spinor_field[NT][NZ][NY][NX];
```

```
void dslash(spinor_field result, su3_field u, spinor_field input)
{
  spinor tmp1, tmp2;

  for(int it=0; it<NT; it++)
    for(int iz=0; iz<NZ; iz++)
      for(int iy=0; iy<NY; iy++)
        for(int ix=0; ix<NX; ix++)
          {
            init_spinor_zero(&tmp2);

            // t-direction, mu=0
            su3_spinor_mul(&tmp1, &(u[it][iz][iy][ix][0]),
                           &(input[INC_T(it)][iz][iy][ix]));
            one_pm_gamma_mul(&tmp2, 0, -1, &tmp1);
            su3_dag_spinor_mul(&tmp1, &(u[it][iz][iy][ix][0]),
                               &(input[DEC_T(it)][iz][iy][ix]));
            one_pm_gamma_mul(&tmp2, 0, 1, &tmp1);

            // x-direction, mu=1
            su3_spinor_mul(&tmp1, &(u[it][iz][iy][ix][1]),
                           &(input[it][iz][iy][INC_X(ix)]));
            one_pm_gamma_mul(&tmp2, 1, -1, &tmp1);
            su3_dag_spinor_mul(&tmp1, &(u[it][iz][iy][ix][1]),
                               &(input[it][iz][iy][DEC_X(ix)]));
            one_pm_gamma_mul(&tmp2, 1, 1, &tmp1);

            // y-direction, mu=2
            su3_spinor_mul(&tmp1, &(u[it][iz][iy][ix][2]),
                           &(input[it][iz][INC_Y(iy)][ix]));
            one_pm_gamma_mul(&tmp2, 2, -1, &tmp1);
            su3_dag_spinor_mul(&tmp1, &(u[it][iz][iy][ix][2]),
                               &(input[it][iz][DEC_Y(iy)][ix]));
            one_pm_gamma_mul(&tmp2, 2, 1, &tmp1);

            // z-direction, mu=3
            su3_spinor_mul(&tmp1, &(u[it][iz][iy][ix][3]),
                           &(input[it][INC_Z(iz)][iy][ix]));
            one_pm_gamma_mul(&tmp2, 3, -1, &tmp1);
            su3_dag_spinor_mul(&tmp1, &(u[it][iz][iy][ix][3]),
                               &(input[it][DEC_Z(iz)][iy][ix]));
            one_pm_gamma_mul(&tmp2, 3, 1, &tmp1);

            result[it][iz][iy][ix] = tmp2;

          }
}
```

```
void mr_solver(spinor_field solution, su3_field u, spinor_field sour
{
  spinor_field  tmp, r, Mr;

  init_spinor_unit(solution);
  m_wilson(tmp, u, solution);
  spinor_field_sub(r, source, tmp);

  double residual = sqrt(norm_square_field(r));

  int count=0;

  complex coef1, coef2;
  double coef;

  while((count<max_iter) && (residual>tolerance))
    {
      ++count;

      m_wilson(Mr, u, r);

      coef1 = spinor_field_prod(Mr, r);
      coef = 1.0/(norm_square_field(Mr));
      coef *= omega;
      coef2 = re_complex_mul(coef1, coef);

      cm_spinor_field_mul(tmp, coef2, r);
      spinor_field_add_assign(solution, tmp);

      cm_spinor_field_mul(tmp, coef2, Mr);
      spinor_field_sub_assign(r, tmp);

      residual = sqrt(norm_square_field(r));

      printf("iteration: %d, residual: %.16e\n", count, residual);

    }

  printf("inversion converge after %d of iterations:\n",count);
  printf("true residual is %.16e\n\n", residual);

}
```

提纲

矩阵 M 的稀疏性质：平铺的时空排列次序



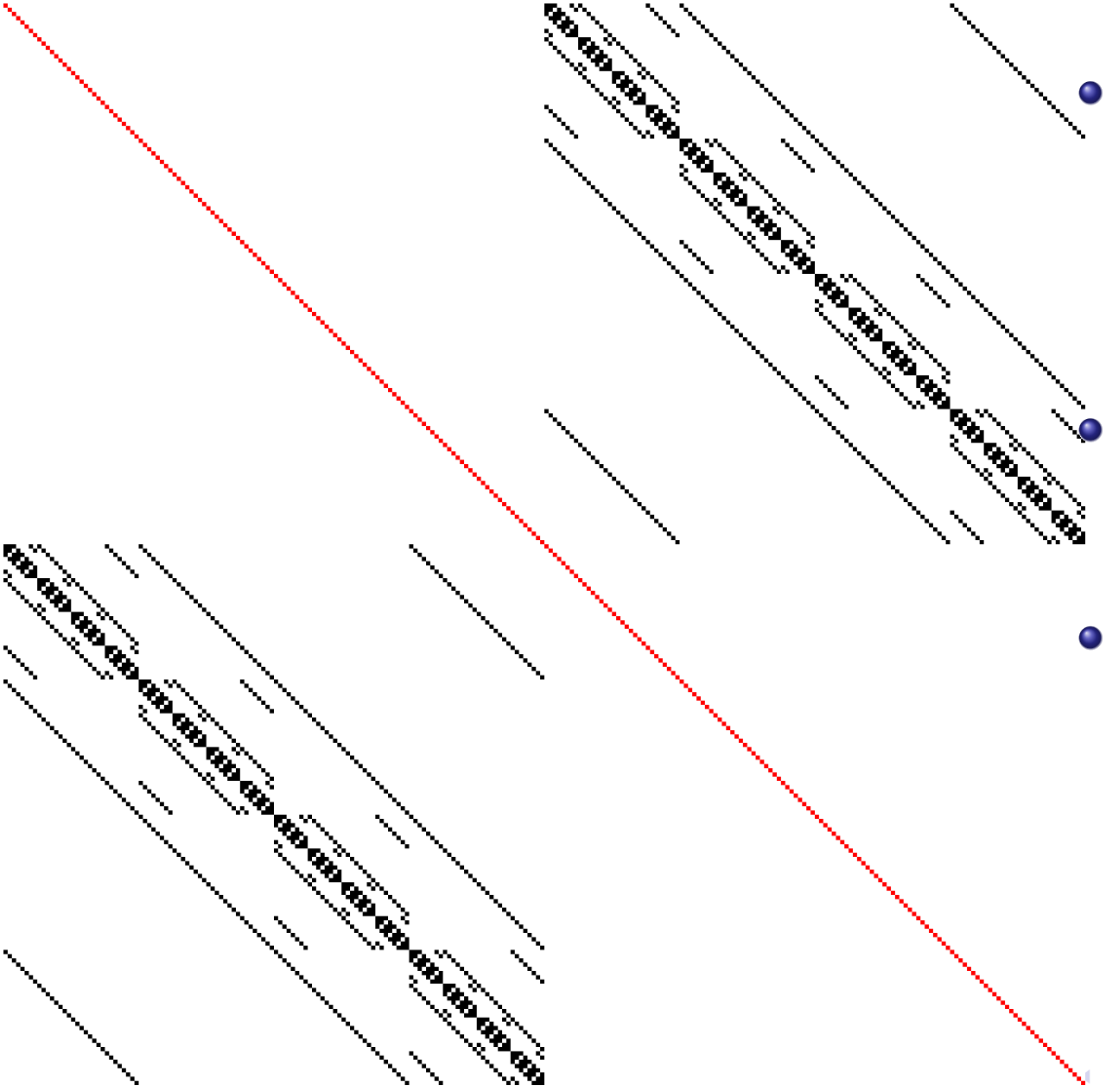
- 图中用 256*256 个点来表示定义在 4^4 时空格子上的 $(4^4 * 12) * (4^4 * 12) = 3072 * 3072$ 矩阵 M。
- 红色点表示 12*12 的对角矩阵，对角元为实常数
- 黑色点表示 12*12 的分块稀疏矩阵，分成 16 个 3*3 的块。一般来说，其中 8 个块是相同或相差符号或 i 的满的复矩阵，另外 8 个块是 0 矩阵。

每个黑色点大约类似是这样（以 γ_2 为例）:

$$\begin{pmatrix}
\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} & \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} & \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix} & \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \\
\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} & \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} & \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} & \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{pmatrix} \\
\begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{pmatrix} & \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} & \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} & \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \\
\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} & \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix} & \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} & \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}
\end{pmatrix}$$

其中 8 个 3*3 块之间或是相同，或是相差一个符号或 i。

矩阵 M 的稀疏性质：奇偶分开的时空排列次序



- 图中用 256*256 个点来表示定义在 4^4 时空格子上的 $(4^4 * 12) * (4^4 * 12) = 3072 * 3072$ 矩阵 M。
- 红色点表示 12*12 的对角矩阵，对角元为实常数
- 黑色点表示 12*12 的分块稀疏矩阵，分成 16 个 3*3 的块。一般来说，其中 8 个块是相同或相差符号或 i 的满的复矩阵，另外 8 个块是 0 矩阵。

“奇-偶”预处理技术

把费米子矩阵 M 做分解

$$
\begin{aligned}
M &= \begin{bmatrix} M_{ee} & M_{eo} \\ M_{oe} & M_{oo} \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 \\ M_{oe}M_{ee}^{-1} & 1 \end{bmatrix} \begin{bmatrix} M_{ee} & 0 \\ 0 & M_{oo} - M_{oe}M_{ee}^{-1}M_{eo} \end{bmatrix} \begin{bmatrix} 1 & M_{ee}^{-1}M_{eo} \\ 0 & 1 \end{bmatrix} \\
&= L\ \tilde{M}\ U
\end{aligned}
$$

- M_{ee} 和 M_{oo} 都是对角的
- M_{oe} 和 M_{eo} 就是把 $\not{\partial}$ 拆成两半

线性方程组 $M\,x = b$ 变为:

$$\tilde{M}(Ux) = L^{-1}b$$

“奇-偶”预处理技术

具体来说：

- 把源向量写成奇偶两部分：

$$b = \begin{bmatrix} b_e \\ b_o \end{bmatrix}$$

- 解新的线性方程：

$$\left(M_{oo} - M_{oe}M_{ee}^{-1}M_{eo}\right)x'_o = \left(b_o - M_{oe}M_{ee}^{-1}b_e\right)$$

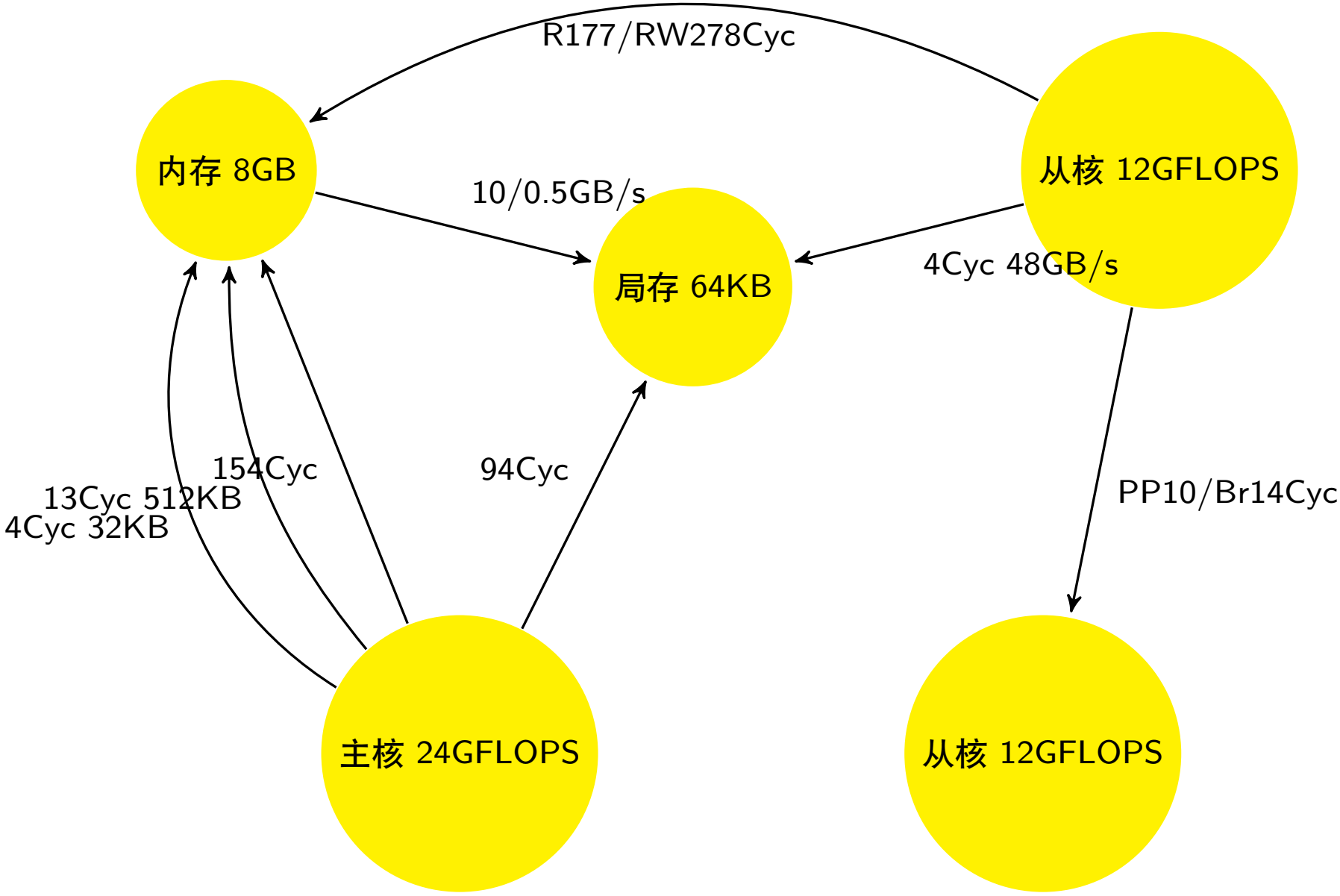
- 组合出原来的解：

$$x = \begin{bmatrix} M_{ee}^{-1}b_e - M_{ee}^{-1}M_{eo}x'_o \\ x'_o \end{bmatrix}$$

关于其他算法

- Krylov 子空间算法有很多种：MR、CG、biCG、biCGstab、GCR 等等，基本思路类似。
- 预处理技术也有很多种，除了简单的“奇-偶”预处理技术外，还有 Multi-Grid、Domain-Decomposition 等。
- 格点 QCD 发展了特有的针对多源同步 (或加速) 求解的改进算法，包括 multi-shift（或称 multi-mass）算法，以及 eigCG、Deflation 算法等。
- 也有很多算法是针对具体的计算环境而设计的，比如 block CG 等。

针对太湖之光的计算环境，也许需要有针对性的设计或改良算法。


R177/RW278Cyc
内存 8GB
从核 12GFLOPS
10/0.5GB/s
局存 64KB
4Cyc 48GB/s
154Cyc
13Cyc 512KB
4Cyc 32KB
94Cyc
PP10/Br14Cyc
主核 24GFLOPS
从核 12GFLOPS

提纲

视角：大概能确定的情况？

- 主核应当负责 MPI 通讯，可能负责边界层的计算。
- 从核是计算的主力，大部分乘法计算以单精度为主（除非发现双精度更快）。
- 代码应充分利用架构的特性，尤其是寄存器通讯等。
- 64KB 的局部内存应该是不够存储所有数据的（只能放大约 3^4 的格子），内存和局存之间的数据流动不可避免。
- 粗略的分析：$\not{D}\, v$ 在一个时空点上约有 1200flop 计算量，却涉及约 1300Byte 的储存量。而太湖之光的从核速度与 DMA 带宽相比，每个 Byte 对应 24flop。所以主要瓶颈是“数据的流速无法喂饱从核的计算能力”。
- 目前国际上在 Intel KNL 处理器上做格点 QCD 计算，最好的计算效率大约是理论峰值的 10% 左右。根据今年 top500 的 HPCG 测试，可以预计在太湖之光上的效率很可能会低于这个值。我们的优势在于规模和 scaling 特性，但也应该尽量做到高效率才好。

视角：主从核分工

- 数据应该主要是在内存和局存间 DMA 传输，但主核是否也应该推送或调取一部分局存数据？这两者是否冲突？
  - 仅有的主核缓存应当被适当利用起来。
- 64 个从核是否是完全对称的？还是有所分工？
  - 考虑到 8 行有 4 个内存控制器，每两行共享一个。如果有部分从核负责搬运数据和/或中转通讯，那么应该设计好任务分配和数据流转的路线。
- 用于计算的从核如何分配任务？
  - 从核排布是二维的，而时空节点是四维的，所以不同的映射方式可能会导致不同的通讯开销。
- $\not{D}$ 的旋量指标部分是否有可能拆分优化并行？
- “流水线”方式对格点 QCD 计算是否有效率？

视角：内存与数据传输

- 为了保证向量化，数据在内存中应当大块读入，按顺序调取，处理时 4 个一组对齐。
  - 是否需要在主内存里同时存放不同排布方式的数据？
- 普适的稀疏矩阵存放方式，如 CSR 等，需要存放非零数据的序号或行列指标。格点 QCD 涉及的数据非常规则，一般可以考虑随时计算位置。
- $SU(3)$ 的 3×3 矩阵可以压缩成 12 个实数或者 8 个实数。这个特性可能会很有用。
- 从核有少量指令 cache，所以有些信息可以 encode 到代码里，而不是放在数据里。
  - 比如数据的位置等可以尽量少用间接寻址。另外，寄存器通讯的逻辑和时序也可以硬编码而不是把路由信息放在数据里。
- 从核的局存之间尽量不存重复的数据，尽量使用寄存器通讯实现跨从核访问。

视角：算法改进

- 求解线性方程组的算法如何改良？
  - 目前有 Domain Decomposition、Multigrid、block CG 等算法或预处理方案值得参考。需要更多灵感……
- 主要问题在于：好不容易传进去的数据，一下子就算完传出来了，处理器一直在等待。
  - 所以，是否能考虑充分利用传进去的数据，多算一些东西出来？
- 局部求逆或矩阵分解，对整体的求解是否有好处？如何利用？
- 反过来看“流水线”方法，可以理解为处理器在格子上沿一个路线跑动，那么是否有“局部更新”的方案来解线性方程组？
- 可否在 $SU(3)$ 群上做些手脚，比如（随机）离散化群空间、按生成元重新排布数据和拆分计算等等脑洞……

视角：最优化理论指导

- 神威架构的编程模型不是单纯的进程（数据传输模型）或线程（共享内存模型），而是某种混合。那么适合它的编程模型是否有发展的可能？
- 针对格点 QCD 这种固定结构的稀疏矩阵，是否可以自动生成逻辑上等价的很多代码，但数据流和任务分配等不同，这样就可以用遗传算法等来寻找最优化代码。这个可行吗？
- 最优代码生成、编译器自动优化等往往受制于程序模型过于灵活，是否可以有针对简单模型（非图灵完全）的优化理论？
- 我在尝试用直接描述 DAG 的语法来表达计算过程，试图用推演 DAG 的基本操作（图的合并、连接等）的办法来表达优化过程。不过我对于这方面理论了解甚少。请求专家指导。

谢谢！