

自由能 *F* = −*k**B**T*

lim

T
↓

{\displaystyle \lim _{T\downarrow }}

 ln*Z*, 磁化强度 *M* = *k**B**T*

∂

ln
Z

∂
H

{\displaystyle {\frac {\partial \ln Z}{\partial H}}}

, 总熵 *U* = −

∂

(

1

k

B
T

)

∂
T

{\displaystyle -{\frac {1}{k_{B}T}}{\partial T}}

 ln*Z*, 比热 *C**V* =

∂

U

∂
T

{\displaystyle {\frac {\partial U}{\partial T}}}

, 磁化率 *χ* =

lim

M
↓

∂
M

∂
H

{\displaystyle \lim _{M\downarrow }{\frac {\partial M}{\partial H}}}

U = ⟨*E*⟩ = Σ*p*_{*α*} *E*_{*α*}, *M* = *N*μ_B⟨*σ*⟩ = μ_B Σ*α* *p*_{*α*} (Σ*σ*_{*α*} = 1), ⟨*E*²⟩ = Σ*p*_{*α*} *E*_{*α*}², *M*² = μ_B² Σ*α* *p*_{*α*} (Σ*σ*_{*α*} = 1)², ⟨Δ*E*²⟩ = ⟨*E*²⟩ − (⟨*E*⟩)², ⟨Δ*M*²⟩ = ⟨*M*²⟩ − (⟨*M*⟩)², *C* =

∂

(

Δ

E

2

)

∂
T

{\displaystyle {\frac {\partial (\Delta E^{2})}{\partial T}}}

, *χ* =

∂

(

Δ

M

2

)

∂
T

{\displaystyle {\frac {\partial (\Delta M)^{2}}{\partial T}}}

一维 Ising 解: *K* =

1

k

B
T

I
=

μ

H

k

B
T

{\displaystyle K={\frac {1}{k_{B}T}}I={\frac {\mu H}{k_{B}T}}}

. 零磁场上, 配分函数*Z* = 2(2cosh*K*)^{*N*−1}. 每个自旋的自由能为 *F* = −*k**B**T*ln(2 cosh *K*), 是温度的解析函数, 没有相变点。有外磁场时, 若采用周期性边界条件, 配分函数*Z* = λ₁^{*N*} + λ₂^{*N*}; λ_± = e^{*K*} cosh l ± √e^{2*K*} sinh 2*l* + e^{−2*K*}. *H* ≠ 0, *F* ≈ ln(e^{*K*} cosh l + √e^{2*K*} sinh 2*l* + e^{−2*K*}), 是温度的解析函数, 没有相变。去掉外磁场,

∂
M

∂
H

= 0,

除了*T* = 0点无自发磁化, 没有有序相变。

二维 Onsager 解: 比热 *C* =

4

k

B

K

2

coth

2

⁡
2*K*
[
K
(
κ
)
−
E
(
κ
)
−
sech

2

⁡
(
2*K*
)
[

z

2

+
(
2
tanh
(
2*K*
)
−
1
)
K
(
κ
)
]
]

{\displaystyle C={\frac {4}{k_{B}}}K^{2}\coth ^{2}\,2K\left[K(\kappa)-E(\kappa)-\operatorname {sech} ^{2}(2K)\left[{\frac {z^{2}}{2}}+(2\tanh(2K)-1)K(\kappa)\right]\right]}

第一类椭圆积分*K*(κ) = ∫₀^{π/2} (1−κ² sin² θ)^{−0.5}dθ 第二类椭圆积分E(κ) = ∫₀^{π/2} (1−κ² sin² θ)^{0.5}dθ

K(κ)在κ = 1处发散, 相变点: *T*_c =

2

ln
⁡

1
1
+
√
κ

≈
2.269

{\displaystyle T_{c}={\frac {2}{\ln 1+{\sqrt {\kappa }}}}\approx 2.269}

三角格子: *T*_c = 3

二维 Ising 规则: 减弱有限边界的影响, 可以假设周期性边界条件。指数的确定同样可以使用有限尺度标度法。

M − (*T*_c − *T*)^β β = 1/8总能图上*T*_c是拐点。

关联长度ξ在相变点处→∞, 意味着每一个自旋都对其他自旋态特别敏感, 因此涨落特别大, 稍加一个外磁场就可以极大地改变体系的磁化强度。

在无外磁场时, *M*的变化是连续的, 为二级相变, 在有限磁场上为一级相变。只是在磁随转变为正值即沿+z轴方向时, 磁化强度突然转向。因此, 磁化强度是不连续变化的, 为二级相变。注意到在 *H* ≈ 0时, *M* 和 *M*[−] 两个状态是等几率的, 但稍加一点外磁场就可使状态取完全不同的几率。在*T*_c ≈ 2.269之下, 都有这种随磁场的不连续变化。但在*T*_c之上, 由于自发磁化消失, 因此当磁场在零值附近变化时磁化强度没有不连续变化行为。由 *M* 由 *H* < 0 的负磁化方向变为 *H* = 0 时的零值, 再继续变为 *H* > 0 时的正值方向。*T*_c 之下的磁化强度的跳跃是自发磁化强度的两倍, 而且在 *T*_c 处自发磁化强度为零, 因此随磁场变化的一级相变与随温度变化的二级相变之间有紧密的关系。低温下存在两个相, 即 ±*m*。在温度高于 *T*_c 之下, 可以通过改变磁场进而不连续地由一个相转变为另一个相。反映在相图上, 有一条一级相变线位于温度轴上, 终止于临界点。在临界温度处, 没有自发磁化, 因此两相的区别消失。等于和高于 *T*_c 处, 可以跨越温度轴而连续地改变 *M*。一级相变终止于临界点是一级相变的普遍特征。

XY 模型: 系统的自旋具有两个任意的取向。E = −*J* Σ*(i,j)* σ_{*i*} · σ_{*j*} = −*J* Σ*(i,j)* σ_{*i*x} σ_{*j*x} + σ_{*i*y} σ_{*j*y} = −*J* Σ*(i,j)* cos(φ_{*i*} − φ_{*j*}),

其中φ = φ_{*i*}为自旋和x轴的夹角。正格子上的二维模型, 对于所有*T* > 0 的温度, 磁化强度*M* = 0, 但在某—温度*T*_{KT}下将发生 Kosterlitz-Touless 相变, 它是一种在无程有序系统中发现的一种拓扑有序。系统可以存在一种亚稳态, 自旋的排列形成涡旋。在相变温度之上, 涡旋是自由的。在相变温度之下, 自旋涡旋是成对出现的, 并且对于 *T* < *T*_{KT} 的所有温度系统都和*T* = *T*_{KT} 时一样, 因此临界点实际上是临界线。计算模拟时, 系统的初始组态首先选取对应于高温 (*T* = ∞) 下的完全无序排列, 即格点上的每个自旋方向是随机选择的, φ = 2π*R*。然后急速降温至*T* > 0, 即改变 Boltzmann 分布中的指数值β = 1/*k**B**T*时步长间隔要大, 在每个温度值下按照上述的 Metropolis 抽样法产生系统的各种构型。由于急速冷却, 系统可以形成长寿的亚稳态涡旋结构。相变温度下正涡旋和反涡旋是同时出现的。

Heisenberg 模型: 自旋可以取三维空间中的任意方向。注意无论是 XY 模型还是 Heisenberg 模型都可以用在一维, 二维, 三维格点上。E = −*J* Σ*(i,j)* σ_{*i*} · σ_{*j*} = −*J* Σ*(i,j)* σ_{*i*x} σ_{*j*x} + σ_{*i*y} σ_{*j*y} + σ_{*i*z} σ_{*j*z} 在系统最高能量的状态即基态上, 所有自旋按完全有序的平行或反平行排列, 在有限的温度下由于热激发出现能量较高的状态。在 Ising 模型中这样的自发激态是自旋的反转, 在 Heisenberg 模型中可以出现周期性的自旋波激发, 它是系统的一种集体行为, 其量子称为磁子。由朗德劳的 Bose 统计可以推出磁化强度与温度*T* 的关系。而在—维 XY 模型中可以出现孤子或孤波激发, 一维链上的自旋发生 2π 的扭转, 在反铁磁情况下这种扭转有 3 种: 一种自旋子格子上的自旋扭转π, 而另一种扭转−π; 两种子格子的自旋各自扭转π; 一种子格子上的自旋不变, 而另一种扭转2π。对于三维 Heisenberg 模型, 其中的一个自旋分量在外加磁场上呈现有序, 而另外两个分量出现类似于 Kosterlitz-Touless 相变的束缚拓扑态激发。随着温度升高, 涡度增加但正负涡旋束缚态开始解离。**q 态 pots 模型**: σ_{*i*}的取值可为*q*个, Hamilton 量E = −*J* Σ*(i,j)* δ_{*σ*_{*i*} σ_{*j*}是, 其中每个格点上的σ_{*i*}取值为 1, 2, ..., *q*。时钟模型: 自旋是个矢量, 它的空间取向限制在二维平面中*q* 个离散指定的方向值的子系统的 Hamilton 量是 E = −*J* Σ*(i,j)* σ_{*i*} · σ_{*j*} *q* > 4 的情形, 是 XY 模型退化成*q* 重向异性的极端情形, 这时系统有两个 Kosterlitz-Touless 相变, 其计算结果的分析十分复杂。}

自旋玻璃: Ising 模型中, 交换常数*J*不再是常数, 可以从高斯分布中选取, 或者赋予正负值不同几率。磁化强度不再是一个好的序参数, 因此用 Edwards-Anderson 参数⟨σ_{*i*}²⟩作为序参数, 其中的(…)是指在某某个初始键分布下的各格点的热平均, 则是指对不同初始键分布*P*(*J*_{*ij*})的平均。**模拟退火法**: 同时模拟不同温度下的系统, 在模拟步骤中途交换这些系统的温度, 因此可以将低温下冻结的状态转移到高温, 以脱离亚稳态。

自旋自相关函数:*C*(*t*) = ∫ *dt*' [m(*t*' + *t*)m(*t*') − (m)²] α exp(−*t*'/*τ*) *m* =

M

N

{\displaystyle m={\frac {M}{N}}}

 (σ = μ)_B *τ*:特征相关时间(与温度有关)**临界慢化**: 在临界点附近, 由于长程关联和无穷大涨落, 采用 Metropolis 算法的单自旋翻转难以实现大集团落下落的集体自旋翻转, 尤其是集团中心自旋翻转概率~ exp(−8/*T*) ≈ 3%, 导致达到近平衡态的计算时间显著延长。

Wolff 算法(自旋簇添加): 1.随机选择一个自旋为自旋簇生长的种子, 检查它的最近邻自旋是否添加到相同; 2.如相符合, 以几率*P*(add)添加到该簇中; 3.对新增自旋的簇成员自旋, 再检查其相邻自旋以确定是否要添加到其中; 4.同样添加新成员时, 以目前未添加到簇的自旋可再被赋予添加新机会; 4.全部添加完成后, 尝试将该簇的自旋集体翻转(翻转几率取于能量势阱); *P*_{add}依赖于温度: 随温度增加而降低。当选择 *P*_{add} = 1 − exp(−2β*J*), 两种翻转都可全部接受, 满足细致平衡条件。

KKKY 相互作用势能: *H* = Σ*(i,j)* *J*(*R*_{*ij*})σ_{*i*} σ_{*j*} *J*(*r*) = *J*₀

cos
⁡

(
2*k**r*π
+
φ
)

(
k

B
T

)

{\displaystyle J(r)={J_{0}}{\frac {\cos \, (2kr\pi +\phi)}{k_{B}T}}}

 长程震荡

Edwards-Anderson 模型:*H* = −Σ*(i,j)* *J*_{*ij*} σ_{*i*} σ_{*j*} *U*_{*i*} = 0 *U*_{*j*} = *J*² *P*(*J*_{*ij*}) =

1

√

2

π

exp
⁡
{
−

β

J

2

}

{\displaystyle P(J_{ij})={\frac {1}{\sqrt {2\pi }}}\exp \left\{-{\frac {\beta J}{\sqrt {2\pi }}}\right\}}

 *p*_{*i*} δ(*J*_{*ij*} − *i*) *C*(*t*) =

1

N

∑

i
=
1

N

(

σ

i
(
t
)

σ

i
(
t
+
0
)
)
−
(

σ

i
)

2

{\displaystyle C(t)={\frac {1}{N}}\sum _{i=1}^{N}(\sigma _{i}(t)\sigma _{i}(t+0))-(\sigma _{i}^{2})}

 序参量:*q* =

1

N

∑

i
=
1

N

(

σ

i
)

2

{\displaystyle q={\frac {1}{N}}\sum _{i=1}^{N}(\sigma _{i}^{2})}

 ≠ 0

Sherrington-Kirkpatrick 模型: *H* = −Σ*(i,j)* *J*_{*ij*} σ_{*i*} σ_{*j*} (*i* ≠ *all*) *U*_{*i*} = 0 *U*_{*j*} = *J*² *P*(*J*_{*ij*}) =

1

√

2

π

exp
⁡
{
−

β

J

2

}

{\displaystyle P(J_{ij})={\frac {1}{\sqrt {2\pi }}}\exp \left\{-{\frac {\beta J}{\sqrt {2\pi }}}\right\}}

或*P*(*J*_{*ij*}) = *p*_{*i*} δ(*J*_{*ij*} − *i*) + *p*_{*i*} δ(*J*_{*ij*} + *i*) (ln*Z*) = ∫ *dJ*_{*ij*} *P*(*J*_{*ij*}) ln*Z*_{*ij*}

副本 (replica) 方法: (ln*Z*) =

lim

N
↓

1

N

(

Z

N

)

{\displaystyle (\ln Z)={\lim _{N\downarrow }{\frac {1}{N}}(Z^{N})}

 (*Z*^{*N*}) = *T**r* exp β Σ*a*−1 Σ*(i,j)* σ_{*i*} σ_{*j*} σ_{*a*} σ_{*a*}

模拟回火法 (Simulated Tempering): 同时模拟两个一样的系统, 其中一个是在高温*T*₁下, 不会产生遍历性破缺; 另一个在低温*T*₀下, 可能处于亚温态的能量势阱中。先用 Metropolis 单自旋翻转方法模拟两个系统各自的演化, 再以一定速率。

交换温度的接受概率:Δ*E* = *E*_{*v*} − *E*_{*w*}

A =

{

1
,
Δ
< 0
exp
⁡
(
−

β

Δ

)
,
Δ
≥ 0

{\displaystyle A={\begin{cases}1,& \Delta < 0 \\ \exp(-\beta \beta \Delta),& \Delta \geq 0\end{cases}}

 低温系统能量较高

遍历性: 高温系统的态直到温度交换时都可遍历, 温度交换后另外一个系统也可遍历, 两个系统是对称的, 因此都具有遍历性。

细致平衡条件: 各自系统的 Metropolis 步选显然是满足的, 交换温度时也是满足的。**KAM 定理**: 一个充分接近可积 Hamilton 系统的不可积系统, 对此系统若把不可积当作可积 Hamilton 函数的扰动来处理, 则在小扰动条件下, 系统运动图像与可积系统基本一致, 当扰动足够大时, 系统图像就发生了性质改变, 成了混沌系统。通俗点说就是, 经典力学的相空间轨迹既不是完全规则, 也不是完全无规, 而是十分敏感地依赖于对起始条件的选择。微小的涨落可能引起混沌的发展。揭示了决定论和随机论之间、牛顿力学和统计力学之间没有不可逾越的界限。

Li-Yorke 定理: 对闭区间 I 上的连续函数 f(x), 如果存在一个周期为 3 的周期点时, 就一定存在任何正整数的周期点, 一定出现混沌现象。

一维迭代 Logistic 方程: *x*_{*n*+1} = λ*x*_{*n*} (1 − *x*_{*n*}) (0 ≤ *x* ≤ 1, 0 ≤ λ ≤ 4)

Feigenbaum 常数: 前后分岔间距的比值(λ_{*m*−1} − λ_{*m*−1})/(λ_{*m*−1} − λ_{*m*}). 趋向于一个常数 δ = 4.669201.

倍周期分岔中的标度行为按以下的几何级数(幂函数)收敛到λ_{*m*} − λ_{*m*} = Δ6^{*m*}. Δ = Δ 是依赖于迭代函数的常数, δ 是不依赖于迭代函数的普适常数。倍周期分岔的图上*x* = 0.5 的直线和放大的分岔曲线相交, 得到分岔纵向往间距*d*₁, *d*₂, *d*₃ 等值。 *d*_{*m*}/*d*_{*m*−1} → α: α是普适常数。α = 2.50290787509589282228390287 混沌的标度: 内随性、分维性、普适性和 Feigenbaum 常数

二维迭代 Hénon 方程: *x*_{*n*+1} = 1 − α*x*_{*n*}² + *y*_{*n*} *y*_{*n*+1} = *b**x*_{*n*} **吸引子**: 轨道在充分长时间之后将渐近地收敛到极限的状态或状态的集合。简单吸引子的维度是整数, 吸引子本身不可分解, 吸引子在相空间的体积为 0。

Lorenz 方程:

d
x

d
t

=
σ
(
y
−
x
)

d
y

d
t

=
r
x
−
x
z
−
y

d
z

d
t

=
x
y
−
b
z

{\displaystyle {\frac {dx}{dt}}=\sigma (y-x){\frac {dy}{dt}}=rx-xz-y{\frac {dz}{dt}}=xy-bz}

奇异吸引子具有分形的性质, 奇异性: 1.对初始条件的尺寸有无限敏感性 2.具有无限嵌套的自相似几何结构, 即具有分维数 3.几何结构完全不随迭代方程的参数而变化而连续地变化, 运动轨迹永远不自我重复, 即具有非周期性。

Rössler 吸引子:

d
x

d
t

=
−
y
−
z

d
y

d
t

=
x
+
a
y

d
z

d
t

=
b
+
z
(
x
−
c
)

{\displaystyle {\frac {dx}{dt}}=-y-z{\frac {dy}{dt}}=x+ay{\frac {dz}{dt}}=b+z(x-c)}

物理系统中的吸引子: 双摆、蔡氏电路、杜芬振荡、杜芬振荡、Gumowski-Mira model **落叶效应**: 当雷诺兹达到 1000 时, 树叶下坠形成混沌运动, 并出现了卡尔曼涡街 **Lyapunov(李雅普诺夫)指数**: 用来表示初值敏感性是否出现以及敏感的程度。经过迭代, 原先很接近的状态之间的差(dx₀, dx₁, dx₂, ...)可能变得很大。dx_{*n*} = dx₀e^{*nλ*}. Lyapunov 指数的'度' =

lim

n
→
∞

ln
⁡

(
d

x

n

)

d

x

0

Δ

x

0

{\displaystyle \lim _{n\rightarrow \infty }{\frac {\ln(d_{n}^{x_{n}})}{\Delta x_{0}}}}

 测量系统动力学特性的一个重要定量指标, 表征了系统在相空间中相邻轨道间收敛或发散的平均指数率, 是对混沌态的初值敏感性的定量判据。正的 Lyapunov 指数可作为混沌行为的判据。系统相空间中运动轨道在每个局部都不稳定、混沌现象、混沌吸引子。Lyapunov 指数小于 0 对应于不动点和各个周期的运动状态。轨道在局部是稳定的; 对初始条件不敏感。 Lyapunov 指数等于零: 对应稳定态轨道, 初始误差不会放大也不会缩小。指数由负变正: 表明周期轨道向混沌的转变。

Julia 集: *Z*_{*n*}+1 = *Z*_{*n*}² + *C* (C = 0: 0和∞是吸引子, 单位圆上的点组成 Julia 集) 多次迭代后的点 *Z*_{*n*} 不趋于某固定点, 也不趋于无穷值时, 则相应的*Z*₀属于该*C* 值下的 Julia 集。实际模拟: 距离函数: *D* = *x*² + *y*²; 每次迭代都计算距离 *D*, 如某初始点*Z*₀(*x*₀, *y*₀), 经过*n*₀次迭代(*n*₀为设置的迭代上限)后仍有*D* < *D*₀ (*D*₀为设定的逃离的边界), 计算点未逃离, 就可以认为该点*Z*₀(*x*₀, *y*₀)属于 Julia 集, 在平面上画出(*x*₀, *y*₀)点; 若还未达到*n*₀ 次迭代, 已经有*D* > *D*₀, 则点*Z*₀不属于该 Julia 集, (*x*₀, *y*₀)点不画出。彩色 Julia 集: 可对每一个*Z*₀, 计算逃离*D* > *D*₀所需迭代的次数*n*, 按逃离所需迭代的次数*n*的不同, 对起始迭代点*Z*₀进行分类, 分别给*Z*₀不同的灰度或颜色, 就可以得到灰度或彩色图案。

Mandelbrot 集: 给定*Z*₀后, 在复参数*C*填充平面上, 对连续*C*进行分类得到图形。通常取*Z*₀ = 0进行迭代, 参数*C*取值, *Z*_{*n*} ≠ 0。对每个*C*(*C* = *a* + *ib*), 以*Z*₀开始, 计算迭代*Z*_{*n*}; 如果当*n* → ∞时, |*Z*_{*n*}| → ∞, 则称*C*(*a*, *b*) 点在*M*集以外, 否则认为*C*点在*M*集内;对每个*C*值, 计算该*C*值下*D*_{*n*} > *D*₀所需的迭代次数*n*, 根据此迭代次数*n*进行分色。**M集的特性**: 自相似性、*M*集概括了所有的*J*集, 是*J*集的缩写字典 广义*J*集, 广义*M*集: *Z*_{*n*+1} = *Z*_{*n*}^{*m*} + *C* (*m* = 3, 4, ...) 拓朴维数: 欧几里得空间的维数与拓朴维数相等 **Hausdorff 维数**: *D* = ln*N*(ε)/ln(1/ε) ε 是测量单元的半径, *N*(ε)是测度得到的规则图形的测量单元数。

Cantor 集: *D* = ln 2 / ln 3 = 0.631 **Cantor 集性质**: 自相似性、无穷操作或迭代过程、精细结构、传统几何陷入危机、长度为 0、简单与复杂的统一 **分形维数均大于拓朴维数**: Koch 曲线: *D* = ln 4 / ln 3 = 1.262; Levy 曲线: *D* = 2; Sierpinski 三角: *D* = ln 3 / ln 2 = 1.585; Sierpinski 正方形: *D* = ln 8 / ln 3 = 1.893; Cisevek 图形: *D* = ln 5 / ln 3 = 1.465; Sierpinski-Menger 海: *D* = ln 20 / ln 3 = 2.777

分形的特点: 图形是“支离破碎的”, 从数学上看它处处是奇点, 如处处不连续或处处不可微; 分形具有标度不变性, 即改变尺度或标度时, 图形是相同的或相似的; 分形的豪斯道夫维数一般是分数(不排除是整数), 并且大于拓朴维数 **布朗运动**: *R*² = *Nb*². *b*²: 位移的均方值 *R*/*b*: 无量纲约化总位移 *N*: 图形包含的单元数, 相当于图形尺寸放大的倍数。 *D* = ln *n* / ln(*R*/*b*) = 2任何维空间中随机行走的维数是 2。 **自回避随机行走**: *D* = (*d* + 2)/3 **粗糙曲线的圆规测量**: 用半径尺寸为 1 的圆规从一端开始作圆弧和海岸线相交, 其交点为下一个圆弧的中心, 这 6 个 1/2 圆, 直到一个分形生长到不同阶段得到的图形进行分维计算, 是将一系列尺寸(α-1)不断增大的方框(也可以是圆) 覆盖到分形图形上, 计数不同方框(或圆)中象素数*N*: *N* ~ *r*^{*D*}. 适用于单个生长图形。对单中心图形, Sandbox 方法更实用。

面积-回转变径法: 把所有分形看成大大小小先后生成的图形*N*−*R*^{*D*} → ln *N* α *D* ln *R*₀

已知分形中心或质心时: *R*_{*i*}² = Σ*r*_{*i*}² /*N* (*i* = 1, 2, 3, ..., *N*) 无明确中心或质心时: *R*_{*0*}² = Σ(*r*_{*i*} − *r*_{*j*})² /2*N* (*N* − 1) (*i*, *j* = 1, 2, 3, ..., *N*) 对包含许多单个圆形的图形, 用面积-回转变径法能得到更准确一些的分维比(从选出几个典型的分形计算的分维值更准确)

变换法: (1)固定矩形宽度*R*₁, 矩形高度由最高和最低点的高度差决定, 一步一步移动矩形, 遍及所有象素点(矩形高度有变化), 将每一象素点处矩形的面积相加, 得到总面积*S*(*R*₁) = Σ*S*_{*i*}(*R*₁, *N*(*R*₁) = *S*(*R*₁)/*R*₁² (2)改变矩形宽度*R*₁ 的大小, 重复以上操作; (3)由*S*(*R*₁)求*N*(*R*₁): 覆盖一部分粗糙曲线所需的面积为*R*²的盒子数, 不过它一般不是整数。 *R*愈大, *N*(*R*)越小; (4)做ln*N*(*R*) − ln(1/*R*)对双数部分图形*N*(*R*−*D* ln*N*(*R*) − ln(1/*R*)变换法计算粗糙曲面的分维计算。(1)此时测量用的矩形被正方柱代替, 正方柱的底面取为 1 × 1, 3 × 3, ..., *R* × *R*。正方柱的高度由正方柱范围内粗糙曲面的最高、最低点高度差决定, 一步一步移动正方柱遍及所有点, 将所有正方柱的体积加起来得到总体积*V*(*R*₁)。 (2)改变*R*₁ 的大小后重复以上操作, 得到一系列体积*V*(*R*₁)。 (3)计算*N*(*R*₁) = *V*(*R*₁)*R*₁³ /*N*(*R*₁)实际上是覆盖一部分粗糙曲面所需的体积为*R*³的盒子数(4)作ln*N*(*R*₁) − ln(1/*R*₁)曲线, 求线性部分斜率*D*。

密度-密度相关函数法: *C*(*r*) =

(

∑

i
=
1

N

ρ

i
(
r
)

ρ

i
(
0
)

)

r
−
α

{\displaystyle C(r)={\frac {(\sum _{i=1}^{N}\rho _{i}(r)\rho _{i}(0))}{r^{-\alpha }}}

 ρ(*r*) = 图形的密度函数, 有图形象素处为 1, 无图形处为 0; *N* 是总象素数; *C*(*r*)的几何意义是: 原始图形和平移 *r* 后的图形重叠部分的象素数和全部象素数的比值, 即在相距 *r* 处发现另一象素的概率; 对于有限的图形, *C*(*r*) 随 *r* 的增大而减小。除了对 *N* 个不同的 *r*² 求平均之外, 还对不同方向、长度相同的 *r* 求平均 特别, 固定*r*², 并把它取为图形的中心, 即*r*² = 0, 此时*C*(*r*) = ρ(0)(*P*(*r*)). ρ表示密度为圆心、*r*为半径的圆周各点上发现另一图形的概率。由于分形具有自相似性, 可以将*C*(*r*)表示为幂函数*C*(*r*)~*r*^{−α}。 *C*(*r*)在回转半径*R*内积分, 在*R*足够大时, 积分值很接近于和图形总象素数*N*成正比, 即∫₀^{*R*} *C*(*r*)*d*²*r* α *N* − *N* α *R*^{α−2} → *D* = *d* − α. *D* 是分形维数, *d* 是欧氏空间维数。

标度不变性: 标度(测量尺度)改变了λ倍后, 函数具有自相似性或标度不变性

本规则: *L* 系统由三部分(*V*, *ω*, *P*)组成。 *V*: 模拟事物的最基本结构或初始结构。 *ω*: 公理, 迭代过程中遵循的一些基本规则。 *P*: 生成规则(或迭代规则)

实际操作: 1、用各种字符符号表示*L* 系统的基本结构*V*, 公理*ω*和迭代规则*P*。 2、按

MPI_INIT_INIT(buf, count, datatype, dest, tag, comm, request)
MPI_SSEND_INIT(buf, count, datatype, dest, tag, comm, request)
MPI_RSEND_INIT(buf, count, datatype, dest, tag, comm, request)
MPI_RECV_INIT(buf, count, datatype, dest, tag, comm, request)
MPI_START(request): 开始某个通信
MPI_STARTALL(count, array_of_requests): 开始所有通信
MPI_REQUEST_FREE(request): 释放某个通信

发送接收和空进程

MPI_SENDRECV(sendbuf, sendcount, sendtype, dest, sendtag, recvbuf, recvcount, recvtype, source, recvtag, comm, status): 发送接收使用不同的缓冲区

MPI_SENDRECV_REPLACE(buf, count, datatype, dest, sendtag, source, recvtag,comm, status):发送接收使用同样缓冲区

MPI_PROG_NULL: 空进程（虚拟进程），对此进程的发送或接收立即返回，在某些情况下编程采用空进程会更加方便

注意：目的地与源、数据个数、数据类型等都可以不一样，可以用普通的MPI_SEND与MPI_RECV来接收，只要匹配集合，没必要非得用MPI_SEDNRECV等

集合通信:所有组内成员间的屏障同步（barrier synchronization）、一个成员到组内所有成员的广播（broadcast）通信
一个成员对所有成员收集（gather）数据、一个成员到组内所有成员分散（scatter）数据、组内所有成员都接收结果（allgather）、组内所有成员到所有成员间的分散/收集数据操作（alltoall）、全局归约（global reduction）操作如求和（sum），求极大值（max），或用用户自定义的操作，结、果返回给所有的组员或仅返回给其中的一个成员、组合归约（combined reduction）和分散操作、组内所有成员上的一组操作（scan）操作（也称前置操作）

注意：对所有集合通信，每个进程都需要执行到这里才能执行，千万不要用进程号或其他方式限制到某个进程来执行，也不需要再采用MPI_SEND和MPI_RECV等来发送接收，否则会导致死锁

数据的打包与解包:一些通信库为发送不连续数据提供打包/解包函数，用户在发送前显式地把数据包装到一个连续的缓冲区中，在接收之后从连续缓冲区中解包。在多数情况下不连续的一个派生数据类型不显式打包和解包，用户指明要发送的和接收的数据的分布，通信库直接访问一个连续的缓冲区。

MPI_PACK(inbuf, incount, datatype, outbuf, outcount, position, comm): 打包

MPI_UNPACK(inbuf, insize, position, outbuf, outcount, datatype, comm): 解包

MPI_PACK_SIZE(incount, datatype, comm, size): 获取打包数据占用空间的上界

一个打包单元可用MPI_PACKED类型发送。任何点数据或收集式通信功能可以用来从一个进程移动构成打包单元的字节序列到另一个进程中。该打包单元这时可用任何接收操作使用任意数据类型来接收；类型匹配的规定对于MPI_PACKED类型发送的消息不再起作用。以任何类型发送的消息（包括MPI_PACKED类型）都可用MPI_PACKED类型接收后被调用MPI_UNPACK来解包。

通信子：将所有这些观点都封装起来以便为MPI中的所有操作提供适当的机会。通信子可分为两种：组内通信子用于一组进程间的操作；组间通信子用于两组进程中的点对点通信。

缓冲区：通信子提供了一个缓冲机制允许人们将与MPI固有特征等价的新的属性联系到通信子上。高级用户可利用这一机制进一步修饰通信子并通过MPI实现一些通信子函数。

组：组定义了一个进程的有序集合，每一进程具有一个序列号，而且为组间通信定义低级名字也是由组完成的。这样组在点对点通信中为进程名字定义了一个范围。另外组还定义了集合操作的范围。在MPI中可从通信子中对组进行分别维护，但是只有通信子才能用于通信操作。

上下文：为MPI提供了拥有分离安全的信息传递空间的能力。

拓扑是加在内部通信子上的额外、可选的属性，它不能被加在组间通信子上。对于一组进程（通信子内部），拓扑能够提供一种方便的命名机制，另外，可以辅助运行时间系统，将进程映射到硬件上。MPI中的进程组是n个进程的集合，组中的每一进程被赋予一个从0到n-1的标识数。在许多并行应用程序中，进程的线性排列不能充分地反映进程间在逻辑上的通信模型（通常由基本问题几何和所用的数字算法所决定），进程经常被排列成二维或三维网络形式的拓扑模型，而且，通常用一个图来描述逻辑进程排列，我们用这种逻辑进程排列为“虚拟拓扑”。在虚拟进程拓扑和底层的物理硬件拓扑之间有一个清晰的差别。进程分配到物理处理器上时，虚拟拓扑可以由系统开发，假设这会帮助提高所给机器的通信性能，然而，这种映射是如何完成的，已超出了MPI的范围。另外，虚拟拓扑的描述仅依赖于应用，并且独立于机器。

MPI环境管理:用于获取和在合适的地方设置关于MPI实现及执行环境（例如错误处理）的不同参数的例程。这里描述了用于进入和离开MPI执行环境的过程。

MPI派生数据类型:想为此止，所有的点对点通信子仅涉及含有相同数据类型相邻缓冲区的数据，对这两种用户限制太大。一种是经常想到送含有不同数据类型值的消息的用户；另一种是经常发送不连续数据的用户。一种解决的办法是在发送端把非连续的数据打包到一个连续的缓冲区中，在接收端再解包。这样做的缺点在于在两端都需要额外的内存到内存拷贝操作，甚至当通信子系统具有分散/分散数据功能的时候也是如此。而MPI提供说明更通用的，混合的非连续通信缓冲区的机制。直到执行（implementation）时再决定数据应该在发送之前打包到连续缓冲中，还是直接从数据库存取。这里提供的通用机制允许不需拷贝，而是直接将各种形式和大小的大小。我们并没有假设MPI能用本地语言描述的连接目标。因此，如用户想要传送一个结构或一个数组部分，则需要向MPI提供一个通信缓冲区的定义，该定义用问题模仿那个结构和数组部分的定义。这些工具可以用于使库设计者定义能够传送用本地语言定义的目标的通信函数：通过对可获得的符号表或虚拟向量（dopevector）的定义解码即可。这种高级通信功能不是MPI的部分。

MPI动态进程管理:组间通信库、动态创建新的MPI进程、独立进程间的通信、基于socket的通信

MPI远程存储访问:远程存储访问即直接对非本地的存储空间进行访问、一个进程对另一个进程的存储区域进行直接访问

MPI并行I/O:显式偏移的并行文件读写、多视口的并行文件并行读写、共享文件读写、分布式数组文件的存取影响MPI程序效率的主要因素:1.并行程序由于进程/线程之间有时候需要相互依赖，某个进程/线程需要等另外的进程/线程完成才可以进行下一步计算，那么会产生等待，导致效率无法100%。2.MPI利用通信进行数据交换，影响MPI程序的主要因素为：网络延迟、网络带宽。

解决措施:
1.根据不同的网络情况决定发送接收的频率及通信包的大小等，比如降低通讯频率，将几次发送接收合并为一次等
2.改进并行模型及算法，有时候比从所用函数等代码方面单纯优化程序对效率影响更大
OpenMP是通过在源代码中添加一些OpenMP编译指令、调用OpenMP库函数来实现共享内存的系统上并行运行的一种标准

OpenMP格式: C/C++中利用pragma预处理指令做为OpenMP的指令，语法格式如下：#pragma omp directive–name [clause[] clause]…] new–line
每个指令以#pragma omp开始，指令其余部分按照C/C++编译指令标准，并区分大小写。#之前和之后可有空白，且有时必须用空白来分隔指令中的文字。#pragma omp之后的预处理目标在编译时将被宏替换。每个OpenMP执行指令必须应用于至少一个随后的语句，并且必须是一个结构块。每行只能有一个OpenMP指令。参数的位置前后之分，可重复。注：[]表示内部的参数是可选的

内在控制变量（ICV）:内置的用于控制OpenMP程序行为的变量，如存储线程数、线程号等。**影响并行块的内在控制变量**: dyn-var: 控制影响的并行区域是否允许动态调整线程数，每任务一个ICV副本 nest-var: 控制影响的并行区域是否允许嵌套并行，每任务一个ICV副本 nthreads-var: 控制影响的并行区域的线程数，每任务一个ICV副

本 thread-limit-var: 控制程序最大允许参与的线程数，整个任务共用一个ICV副本 max-active-levels-var: 控制嵌套线程嵌套的规则如下:1.工作共享区域：可没被嵌套封闭在工作共享、显式 task、critical、ordered 或 master 区域 2.barrier 区域：可没被嵌套封闭在工作共享、显式 task、critical、ordered 或 master 区域 3.master 区域：可没被嵌套封闭在工作共享或显式 task 区域 4.ordered 区域：可没被嵌套封闭在 critical 或显式 task 区域 5.ordered 区域：必须被嵌套封闭在具有 ordered 参数的循环区域或并行循环区域中 6.critical 区域：可没被嵌套（封闭或其它）在具有同样名字的 critical 区域中，此限制并不充分能防止一个死锁
单名字程序
omp_init_(nest)_lock 分别初始化单锁和嵌套锁 void omp_init_(nest)_lock(omp_(nest)_lock_t *lock);
omp_destroy_(nest)_lock 分别销毁单锁和嵌套锁 void omp_destroy_(nest)_lock(omp_(nest)_lock_t *lock);
omp_set_(nest)_lock 分别设置单锁和嵌套锁 void omp_set_(nest)_lock(omp_(nest)_lock_t *lock);
omp_unset_(nest)_lock 分别复位单锁和嵌套锁 void omp_unset_(nest)_lock(omp_(nest)_lock_t *lock);
omp_test_(nest)_lock 分别测试单锁和嵌套锁，如存在则设置 int omp_test_(nest)_lock(omp_(nest)_lock_t *lock);
获取计时器相关变量:

omp_get_wtime 获取以秒为单位的当前时间 double omp_get_wtime(void);
omp_get_wtick 获取计时器的精度 double omp_get_wtick(void);
利用时间程序可以计算程序中某段计算花费的时间
double start, end;
start = omp_get_wtime();
... work to be timed ...
end = omp_get_wtime();
printf("Work took %f seconds\n", end - start);
MPI:进程级、分布式内存、显式、可扩展性好
OpenMP: 线程级（并行粒度）、共享内存、隐式（数据分配方式）、可扩展性差

OpenMP采用共享内存，只适应SMP、DSM架构，不适合集群架构
MPI适合于广泛架构，但编程模型复杂：1.需要分析及划分应用程序问题，并将问题映射到分布式进程集合 2.需要解决通信延迟大和负载不平衡两个主要问题 3.调试麻烦 4.程序可靠性差，一个进程出问题，整个程序将错误
影响 OpenMP 并行程序的因素:OpenMP 程序为共享内存的，各线程可以直接读写其它线程的内存，无需通过网络来进行数据交换：1.避免频繁设置内存私有或共享 2.降低写同样内存的操作，为了保证正确，各线程写同样内存时，需要排队，导致效率降低

常见数值函数库:
一般数值函数库:基本线性代数库(BLAS)、线性代数库(LAPACK)、可扩展线性代数库(SciLAPACK)、傅立叶变换程序(Fourier Transform functions, FFT)、Netlib(http://www.netlib.org): 收集了大量常见的数学函数库
数值函数库的集合名:MKL: Intel Math Kernel Library、AMD CPU 库、IMSL: Rogue Wave International Mathematics and Statistics Library
作业处理、计算可视化软件:Origin、Mathematica、Maple、MATLAB、IDL、GNUPLLOT、Python
作业调度系统:在一个大型系统中，通常需要处理一些自动化运行的任务，通常会采用系统自带的 crontable 的定时任务完成。但是，很多情况下，是多个作业，彼此先后执行，共同完成任务。在这样的情况下，定时任务存在两个明显的问题：1.浪费大量的系统等待时间 2.假设两个作业，第一个作业必须在第二个作业前运行，如第二个作业先运行，就会有灾难性的后果，对于定时任务而言，解决任务作业优先级的问题只是只能把任务一的运行时间安排在二之前，不能完全满足前面的假设，但是对于作业调度器而言，安排作业的优先级，是最基本的功能，简直是太小Case。

常见作业调度系统:LSF(IBM Platform Load Sharing Facility)、SLURM(Simple Linux Utility for Resource Management)、PBS(PBS Pro、OpenPBS、TORQUE)、Maui、Condor、SGE(Oracle Grid Engine)、LoadLeveler(IBM Tivoli Workload Scheduler)
#include "mpi.h"
int main(argc,argv)
int argc; char *argv[];
{
double sum, sum_local;
double a[256], b[256];
int i, n, numprocs, myid, my_first, my_last;
n = 256;
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &numprocs); /*注意&*/
MPI_Comm_rank(MPI_COMM_WORLD, &myid);
my_first = myid * n / numprocs; /*利用序号及进程数使得每个进程的my_first+my_last不一样*/
my_last = (myid + 1) * n / numprocs;
for (i = 0; i < n; i++) {
a[i] = i * 0.5; b[i] = i * 2.0;
}
sum_local = 0;
for (i = my_first; i < my_last; i++) { /*每个进程的my_first+my_last不一样，*/
sum_local = sum_local + a[i] * b[i]; /*每个进程计算for循环的不同部分实现并行*/
}
MPI_Allreduce(&sum_local, &sum, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);
if (myid == 0) printf("%sum=%f\n", sum);
MPI_Finalize();
}
int main(argc,argv)
int argc;
char *argv[];
{
double sum;
double a[256], b[256];
int i;
int status;
int i, n=256;
for (i = 0; i < n; i++) {
a[i] = i * 0.5;
b[i] = i * 2.0;
}
sum = 0;
#pragma omp parallel for reduction(+:sum) /*仅需该代码即完成OpenMP并行*/
for (i = 1; i <= n; i++) {
sum = sum + a[i]*b[i];
}
printf ("%sum=%f\n", sum);
}

```
#pragma omp parallel [clause[ ] clause] ... ] new–line \n structured–block
#pragma omp sections [clause[ ] clause] ... ] new–line [ #pragma omp section new–line] \n structured–block \n
[ #pragma omp section new–line \n structured–block \n ... ]
单执行结构指明相关代码只能有一个线程（并不必须是雇主线程， 0号线程）执行，一般为先到的线程执行
#pragma omp single [clause[ ] clause] ... ] new–line \n structured–block
工作共享结构(workshare Construct)指明相关的代码被分割成不同的单元以供不同线程执行，不同单元只能被某个线程执行一次
C/C++：不支持此OpenMP指令
联合并行工作共享结构实际上是在并行结构中嵌套共享结构的一个快捷方式，这些指令将同时指明一个工作。共享结构具有并行性，而不需其它语句另行指明。允许一些并行结构和工作共享结构都允许特定参数。如程序含有对并行结构或工作共享结构具有不同行为的参数，那么其行为将不可预测。
并行循环结构指明一个或多个相关的循环迭代将被一组线程并行执行
#pragma omp parallel for [clause[ ] clause] ... ] new–line \n for–loops
并行分块结构指明与之相关的不同区域的代码将被不同线程并行执行
#pragma omp parallel sections [clause[ ] clause] ... ] new–line [ #pragma omp sction new–line] \n structured–block \n
[ #pragma omp section new–line \n structured–block] \n ... ]
并行工作共享结构指明相关的代码被分割成不同的单元以供不同线程并行执行,不同单元只能被某一线程执行一次
C/C++：不支持此OpenMP指令
任务结构定义一个明确的任务
#pragma omp task [clause[ ] clause] ... ] new–line \n structured–block
雇主结构指明结构块需要雇主线程（主线程， 0号线程）执行
#pragma omp master new–line \n structured–block
临界结构指明结构块在同一时间只能有一个线程执行，一个线程执行完毕，其它线程才能执行该处代码
#pragma omp critical [name] new–line \n structured–block
栅栏结构指明在此处有一个栅栏，需进行显式同步，即需本组内所有线程都要运行到此后才执行后续代码(栅栏结构必须在所有线程的同样顺序中)
#pragma omp barrier new–line
任务等待结构指明需要自当前任务开始就需等待产生的子任务完成
#pragma omp taskwait newline
```

原子结构用于确保指明的存储区域对某项操作进行原子更新，而不使其暴露给多个同步进行写的线程，避免同时多个线程对这些变量进行更新

#pragma omp atomic new–line \n expression–stmt

刷新结构指明对指定的变量执行内存刷新操作以保证线程此时看到这些变量在内存中的值一致

#pragma omp flush [list] new–line

有序结构指明循环结构中结构块的执行按照循环的迭代顺序，将排序此结构块中的执行顺序，并允许结构块之外的代码可以并行执行

#pragma omp ordered new–line \n structured–block

结构内引用的变量的数据共享属性法则:结构中引用变量的数据共享属性可为预先决定的、显示决定的或隐式决定的之一。封闭结构中的firstprivate、lastprivate 或 reduction 参数声明的变量会导致此结构中变量采用隐式的引用方式，并遵守以下规则：threadprivate 指令指定的变量是线程私有的、结构中某个范围内声明的具有自动存储周期的变量是私有的、具有堆栈分配存储的变量是共享的、静态数据成员是共享的、for 循环或 parallel 对循环结构中的循环迭代变量是私有的、不具有易变成员的保留常数变量是共享的、结构中某个范围内声明的静态变量是共享的。除了在以下情形中，具有预定义数据共享属性的变量也许未在该数据共享属性参数中列出，对这些例外，在数据共享属性参数中允许列出一个预先定义的变量，且将取这些变量的预先定义数据共享属性: 对循环或 parallel 对循环结构中的循环迭代变量可在 private 或 lastprivate 参数中被列出。
数据共享法则:具有显式决定或隐式决定的数据共享属性的变量：1.具有显式决定的数据共享属性的变量：在一个给定的结构中被引用并且在此结构的数据共享参数中列出的变量 2.具有隐式决定的数据共享属性的变量：在一个给定的结构中被引用，但没有被预先决定数据共享属性，且没有在此结构的数据共享属性参数中列出的变量。
隐式数据共享法则:1.在 parallel 或 task 结构中，如存在 default 参数，那么这些变量的数据共享属性由 default 参数决定 2.在 parallel 结构中，如没有 default 参数，那么这些变量是共享的 3.对于不是 task 的结构来说，如没有 default 参数，那么这些变量将从封闭上下文继承数据共享属性 4.在 task 结构中，如没有 default 参数，那么在所有封闭结构并一直到最内部的封闭 parallel 结构中变量被决定为共享的 5.在 task 结构中，如没有 default 参数且数据共享属性没有被上述规则决定，那么变量是 firstprivate 的。
区域而非结构中的数据共享属性规则:1.在被调用子程序中声明的静态变量在此区域中是共享的 2.不具有易变成员且在被调用子程序中声明的保留常数变量是共享的 3.除非是在 threadprivate 指令中出现，否则区域中在被调用子程序中引用的文件范围或者名字空间范围变量是共享的 4.具有堆栈分配存储的变量是共享的 5.除非是在 threadprivate 指令中出现，否则静态数据成员是共享的 6.区域中被调用子程序的通过引用传递的正式参数继承关联的实际参数数据的数据共享属性 7.区域中被调用子程序中的其它变量是私有的

threadprivate 指令指明变量是线程私有的

#pragma omp threadprivate(list) new–line

default: 设置默认共享方式

default(shared | none)

其中：shared: 线程共享

reduction: 声明对变量进行规约操作

reduction(operator:list)

数据复制参数 copyin(list):在执行 parallel 结构时,将主线程的线程私有变量的值复制给所有其它线程
copyprivate(list):提供一种机制可以将隐式任务数据环境的某个私有变量的值广播给此 parallel 结构中的其它数据环境

控制嵌套线程嵌套的规则如下:1.工作共享区域：可没被嵌套封闭在工作共享、显式 task、critical、ordered 或 master 区域 2.barrier 区域：可没被嵌套封闭在工作共享、显式 task、critical、ordered 或 master 区域 3.master 区域：可没被嵌套封闭在工作共享或显式 task 区域 4.ordered 区域：可没被嵌套封闭在 critical 或显式 task 区域 5.ordered 区域：必须被嵌套封闭在具有 ordered 参数的循环区域或并行循环区域中 6.critical 区域：可没被嵌套（封闭或其它）在具有同样名字的 critical 区域中，此限制并不充分能防止一个死锁

单名字程序
omp_init_(nest)_lock 分别初始化单锁和嵌套锁 void omp_init_(nest)_lock(omp_(nest)_lock_t *lock);
omp_destroy_(nest)_lock 分别销毁单锁和嵌套锁 void omp_destroy_(nest)_lock(omp_(nest)_lock_t *lock);
omp_set_(nest)_lock 分别设置单锁和嵌套锁 void omp_set_(nest)_lock(omp_(nest)_lock_t *lock);
omp_unset_(nest)_lock 分别复位单锁和嵌套锁 void omp_unset_(nest)_lock(omp_(nest)_lock_t *lock);
omp_test_(nest)_lock 分别测试单锁和嵌套锁，如存在则设置 int omp_test_(nest)_lock(omp_(nest)_lock_t *lock);
获取计时器相关变量:

omp_get_wtime 获取以秒为单位的当前时间 double omp_get_wtime(void);

omp_get_wtick 获取计时器的精度 double omp_get_wtick(void);

利用时间程序可以计算程序中某段计算花费的时间

double start, end;

start = omp_get_wtime();

... work to be timed ...

end = omp_get_wtime();

printf("Work took %f seconds\n", end - start);

MPI:进程级、分布式内存、显式、可扩展性好

OpenMP: 线程级（并行粒度）、共享内存、隐式（数据分配方式）、可扩展性差

OpenMP采用共享内存，只适应SMP、DSM架构，不适合集群架构

MPI适合于广泛架构，但编程模型复杂：1.需要分析及划分应用程序问题，并将问题映射到分布式进程集合 2.需要解决通信延迟大和负载不平衡两个主要问题 3.调试麻烦 4.程序可靠性差，一个进程出问题，整个程序将错误
影响 OpenMP 并行程序的因素:OpenMP 程序为共享内存的，各线程可以直接读写其它线程的内存，无需通过网络来进行数据交换：1.避免频繁设置内存私有或共享 2.降低写同样内存的操作，为了保证正确，各线程写同样内存时，需要排队，导致效率降低

常见数值函数库:

一般数值函数库:基本线性代数库(BLAS)、线性代数库(LAPACK)、可扩展线性代数库(SciLAPACK)、傅立叶变换程序(Fourier Transform functions, FFT)、Netlib(http://www.netlib.org): 收集了大量常见的数学函数库
数值函数库的集合名:MKL: Intel Math Kernel Library、AMD CPU 库、IMSL: Rogue Wave International Mathematics and Statistics Library
作业处理、计算可视化软件:Origin、Mathematica、Maple、MATLAB、IDL、GNUPLLOT、Python
作业调度系统:在一个大型系统中，通常需要处理一些自动化运行的任务，通常会采用系统自带的 crontable 的定时任务完成。但是，很多情况下，是多个作业，彼此先后执行，共同完成任务。在这样的情况下，定时任务存在两个明显的问题：1.浪费大量的系统等待时间 2.假设两个作业，第一个作业必须在第二个作业前运行，如第二个作业先运行，就会有灾难性的后果，对于定时任务而言，解决任务作业优先级的问题只是只能把任务一的运行时间安排在二之前，不能完全满足前面的假设，但是对于作业调度器而言，安排作业的优先级，是最基本的功能，简直是太小Case。

常见作业调度系统:LSF(IBM Platform Load Sharing Facility)、SLURM(Simple Linux Utility for Resource Management)、PBS(PBS Pro、OpenPBS、TORQUE)、Maui、Condor、SGE(Oracle Grid Engine)、LoadLeveler(IBM Tivoli Workload Scheduler)
#include "mpi.h"
int main(argc,argv)
int argc; char *argv[];
{
double sum, sum_local;
double a[256], b[256];
int i, n, numprocs, myid, my_first, my_last;
n = 256;
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &numprocs); /*注意&*/
MPI_Comm_rank(MPI_COMM_WORLD, &myid);
my_first = myid * n / numprocs; /*利用序号及进程数使得每个进程的my_first+my_last不一样*/
my_last = (myid + 1) * n / numprocs;
for (i = 0; i < n; i++) {
a[i] = i * 0.5; b[i] = i * 2.0;
}
sum_local = 0;
for (i = my_first; i < my_last; i++) { /*每个进程的my_first+my_last不一样，*/
sum_local = sum_local + a[i] * b[i]; /*每个进程计算for循环的不同部分实现并行*/
}
MPI_Allreduce(&sum_local, &sum, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);
if (myid == 0) printf("%sum=%f\n", sum);
MPI_Finalize();
}
int main(argc,argv)
int argc;
char *argv[];
{
double sum;
double a[256], b[256];
int i;
int status;
int i, n=256;
for (i = 0; i < n; i++) {
a[i] = i * 0.5;
b[i] = i * 2.0;
}
sum = 0;
#pragma omp parallel for reduction(+:sum) /*仅需该代码即完成OpenMP并行*/
for (i = 1; i <= n; i++) {
sum = sum + a[i]*b[i];
}
printf ("%sum=%f\n", sum);
}

TEX/LATEX:优点:1.排版的效果非常整齐漂亮 2.排版的效率非常高 3.非常稳定，从1995年到现在，TEX系统只发现过一个bug 4.排版科技文献，尤其是含有很多数学公式的文獻特别方便、高效。现今没有一个排版软件在排版数学公式上面能和

TEX/LATEX相媲美 5.纯文本，可用任何编辑器编辑；缺点：不是

WYSIWYG，需编译后才知效果