

# 基于 Intel Xeon/Xeon Phi 平台的关于离散时间对冲误差的并行化研究

叶帆<sup>1,2</sup> 陈浪石<sup>1,3</sup> 潘慈辉<sup>1,4</sup>

<sup>1</sup>Maison de la Simulation

<sup>2</sup>Atomic Energy and Alternative Energies Commission (C.E.A)

<sup>3</sup>French National Centre for Scientific Research (C.N.R.S)

<sup>4</sup>Versailles Saint-Quentin-en-Yvelines University



November 16, 2014

# 内容提要

- 1 背景介绍
- 2 课题陈述
- 3 并行化算法
- 4 实验结果及分析

# 内容提要

- 1 背景介绍
- 2 课题陈述
- 3 并行化算法
- 4 实验结果及分析

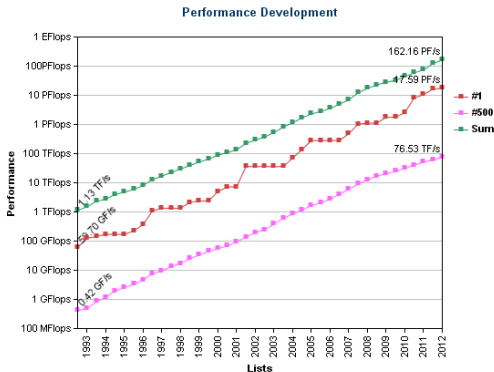
# 超算发展及 Xeon Phi

高性能计算已经进入后 Petaflop( $10^{15}$ )时代。但是能优化到 Petaflop 级别的实际应用却很少，面临问题如下

- 算法的内在并行性不足
- 并行算法的通信受制于内存和网络连接
- 并行编程的难度

超级计算机本身还面临

- 能耗过大



超算 Top500 发展趋势

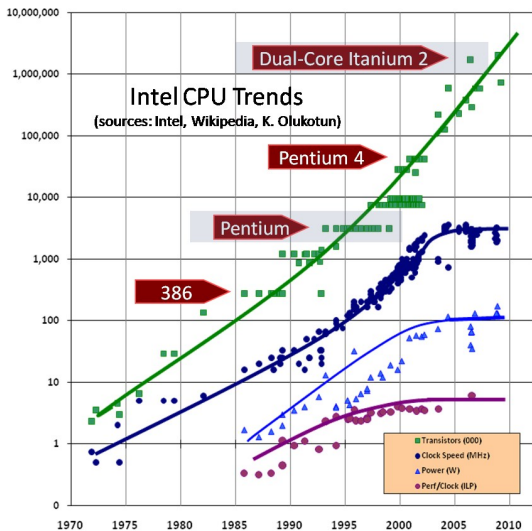
# 超算发展及 Xeon Phi

单个 CPU 的发展已经达到性能和功耗的瓶颈，超算转向众核架构(Manycore)，其具有如下优势

- 高带宽(bandwidth)带来的高通量(high data throughput)
- 高能耗效率(flops/watt)
- 适合大规模的数据并行性应用(data parallel)

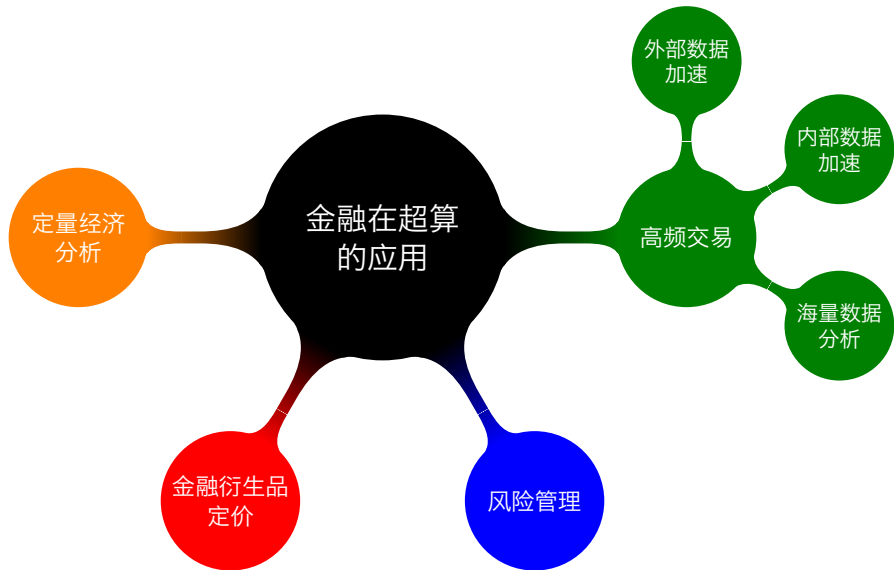
目前众核结构处理器的代表

- Nvidia 的 GPU 加速器
- Intel 的 Xeon Phi 协处理器



单个 CPU 发展遇到的各种瓶颈问题

# 金融领域的超算



金融在高性能计算上的几大应用

# 金融领域的超算

## 高频交易

从那些人们无法利用的极为短暂的市场变化中寻求获利的计算机化交易

- ① 数据在交易所和计算机之间加速(co-location)
- ② 数据在计算机内部加速(网卡+CPU 或者 FPGA)
- ③ 海量数据的分析工作

## 金融衍生品定价

衍生品的复制与对冲策略，以减少最终收益的不确定性

- 模型的复杂度带来了密集的计算量
- 对冲的时效性需要计算的高速性

金融交易越来越依赖计算机程序和高性能计算集群

# 内容提要

- 1 背景介绍
- 2 课题陈述
- 3 并行化算法
- 4 实验结果及分析



# Black-Scholes 模型

经济金融中的离散时间模型和连续时间模型之间的关系被广泛的研究。在不完备的市场中，经费自给的交易策略是不可能完美的复制连续时间模型中的金融衍生证券。我们希望研究离散化的交易策略和连续时间的交易策略的误差。我们的研究基于 Black-Scholes 模型。

广义 Black-Scholes 模型:

- 风险资产:  $dX_t = \mu(X_t)X_t dt + \sigma(X_t)X_t dB_t$ ,  $X_0 > 0$
- 无风险资产:  $dS_t^0 = rS_t^0 dt$
- 中性风险概率:  $W_t = B_t + \int_0^t \frac{\mu(X_s) - r}{\sigma(X_s)} ds$  为布朗运动.
- 回报函数:  $f \in L^2(X_t)$ .

简单起见, 我们假设  $r = 0$ ,  $\sigma(X_t) = \sigma$ ,  $f(x) = (x - K)^+$

## 交易误差

另  $u(t, x) = \mathbb{E}(e^{-r(T-t)} f(X_T^x | \mathcal{F}_t))$ , 期权定价公式为:

$$e^{-rT} f(X_T) = h(f) + \int_0^T \xi_t d\tilde{X}_t$$

- 期权价格:  $h(f) = u(0, x)$
- delta 对冲策略:  $\xi_t = \frac{\partial u}{\partial x}$

连续时间的 delta 对冲策略和离散时间的 delta 对冲策略所产生的差:

$$\begin{aligned} M_T^N &= e^{-rT} f(X_T) - (u(0, x) + \int_0^T \frac{\partial u}{\partial x}(\varphi(t), X_{\varphi(t)}) d\tilde{X}_t) \\ &= \int_0^T \frac{\partial u}{\partial x}(t, X_t) d\tilde{X}_t - \int_0^T \frac{\partial u}{\partial x}(\varphi(t), X_{\varphi(t)}) d\tilde{X}_t \end{aligned}$$

## 交易误差

$$\begin{aligned}M_T^N &= (X_T - K)^+ - \mathbb{E}[(X_T - K)^+] - \int_0^T \frac{\partial u}{\partial x}(\varphi(t), X_{\varphi(t)}) dX_t \\&= -\frac{X_0}{\sqrt{2\pi}} \int_{-\infty}^{\frac{\log(\frac{X_0}{K}) + \frac{1}{2}\sigma^2 T}{\sigma\sqrt{T}}} e^{-\frac{v^2}{2}} dv + \frac{K}{\sqrt{2\pi}} \int_{-\infty}^{\frac{\log(\frac{X_0}{K}) - \frac{1}{2}\sigma^2 T}{\sigma\sqrt{T}}} e^{-\frac{v^2}{2}} dv \\&\quad + (X_T - K)^+ - \sum_{i=0}^{n-1} \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\frac{\log(\frac{X_{t_i}}{K}) + \frac{1}{2}\sigma^2(T-t_i)}{\sigma\sqrt{T-t_i}}} e^{-\frac{v^2}{2}} dv (X_{t_{i+1}} - X_{t_i})\end{aligned}$$

这里  $t_k = kT/N$ ,  $k \in \{0, \dots, N\}$  以及  $\varphi(t) = \sup\{t_i | t_i \leq t\}$ .

# 串行算法

---

**Step1:** 给定  $M$  (蒙特卡洛模拟的次数) 和  $N$  (离散时间点的数目), 生成矩阵  $M \times N$  的  $NRV[M][N]$ . 该矩阵由  $M \times N$  个独立标准正态分布的随机变量组成。

**Step2:** 生成布朗运动矩阵  $BM[M][N]$ :  $BM[i][j] = BM[i][j-1] + \sqrt{\frac{T}{N}}NRV[i][j]$  初始条件为  $BM[i][1] = \sqrt{\frac{T}{N}}NRV[i][1]$ ,  $1 \leq i \leq M, 1 \leq j \leq N$ .

**Step3:** 生成价格矩阵  $PX[M][N]$ :  $PX[i][j] = x_0 \exp(-\frac{1}{2}\sigma^2 j \frac{T}{N} + \sigma BM[i][j])$ ,  $1 \leq i \leq M, 1 \leq j \leq N$ .

**Step4:** 对每个  $1 \leq i \leq M$ , 计算  $M_T^N[i]$ . 计算  $M_T^N[i]$  小于  $\epsilon$  的次数  $s$ , 计算模拟仿真的概率  $P = \frac{s}{M}$ .

**Step5:** 如果  $P$  大于阈值概率  $Prob$ , 则我们降低  $N$ , 反之增加  $N$ .

**Step6:** 由新给出的  $N$  重复上述过程.

---

## 参数选择

我们进行  $M$  次蒙特卡罗模拟实验，对于每一次模拟，选取某个  $N$ ，计算相应的误差  $M_T^N$ ，我们希望该误差小于可接受的最大误差  $\epsilon = 10^{-2}$ 。我们用蒙特卡洛的方法计算  $M_T^N \leq \epsilon$  的概率，并希望该概率能超过某一个特定值： $Prob = 95\%$ 。在这个问题中  $M$  和  $N$  的初始值以及其更新方法至关重要。对  $M$  而言，一方面较大的  $M$  会增大蒙特卡罗方法以及通过此方法计算出来的概率的可信度，另一方面  $M$  过大，将不可避免的造成更高的资源和硬件的要求，如何在保持一定可行度的条件下选择一个较小的  $M$  需要一个精巧的方法。对  $N$  而言，较大的  $N$  会使得逼近更准确，从而误差  $M_T^N$  较小，但也会造成资源的浪费和迭代收敛的过程冗长。较大或者较小的  $M$  和  $N$  的初始值将导致收敛缓慢甚至不收敛。如何选择合适的  $M$  和  $N$  的初始值，如何让它们互相迭代更新，以及怎样判断收敛与否是本章所需要解决的问题。关于  $N$  的初始值的选取，一个首要的问题是，数值模拟中， $M$  会影响我们对  $N$  的判断，甚至一个选取不好的  $M$  会给出错误的  $N$  的更新方向，从而导致问题得不到收敛。为了避免这种问题，我们在这里给出一种不依赖于  $M$  的  $N$  的初始值选定方法。

## 参数选择

我们需要一个尽可能小的  $N$ ，使得存在某个关于  $N$  的函数  $P(N)$ ，如下不等式成立：

$$\text{Probability}(M_T^N \geq \epsilon) = \mathbb{P}(M_T^N \geq \epsilon) < P(N) \quad (1)$$

若我们能发现这样的函数  $P(N)$ ，则所有满足不等式：

$$P(N) \leq 1 - \text{Prob} = 5\% \quad (2)$$

的正整数  $N$ ，都满足  $\mathbb{P}(M_T^N < \epsilon) > \text{Prob} = 95\%$ 。我们只需要取满足不等式的最小的正整数  $N$  即可。此时  $N$  的选取独立于  $M$ ，也即无论我们选取怎样的  $M$ ，理论上都会有  $\mathbb{P}(M_T^N < \epsilon) > \text{Prob}$  成立。

## 参数选择

我们的研究推导给出了  $N$  的估界:

### Theorem

假设如上文, 我们有:

$$\mathbb{P}(M_T^n \geq \gamma) \leq \begin{cases} \frac{128\sigma^{\frac{1}{2}}x_0^2T^{\frac{5}{4}}e^{\sigma^2}}{e^{\frac{1}{4}}\gamma^2\pi(\log(\frac{x_0}{K})+\frac{1}{2}\sigma^2T)^{\frac{1}{2}}(2\pi)^{\frac{1}{4}}}\frac{1}{n}, & \log(C_1n) < 3\log(2) + 3 \\ T^{\frac{1}{4}}\left(\frac{\sigma}{(\log(\frac{x_0}{K})+\frac{1}{2}\sigma^2T)\sqrt{2\pi}}\right)^{\frac{1}{2}}e^{-\frac{1}{4}}e^{-\frac{3\gamma^{\frac{2}{3}}\pi^{\frac{1}{3}}}{2ex_0^{\frac{2}{3}}T^{\frac{1}{3}}16^{\frac{1}{3}}e^{\frac{\sigma^2}{3}}}n^{\frac{1}{3}}}, & \log(C_1n) \geq 3\log(2) + 3 \end{cases} \quad (3)$$

## 参数迭代

在由上面定理确定了  $N$  的迭代初始值之后  $N_0$ ，我们可以使用该值来确定一个合适大小的  $M$  的迭代初始值。一种选取方式是在  $N$  为其初始迭代值时，选取尽量小的  $M$  的值应该使得蒙特卡罗模拟出的概率  $\mathbb{P}(M_T^N < \epsilon)$  收敛稳定。也即，固定  $N$  为其初始值不变，当  $M$  在其迭代初始值附近扰动时，相应的  $\mathbb{P}(M_T^N < \epsilon)$  也只是在可接受范围内微小扰动。

在我们的问题背景下，通过动态调优和经验发现，一个较为恰当的  $M$  初始值可以为  $M = 10^5$ ，我们通过比较  $M = 10^5 + i * 1000$ ,  $i = 0, \dots, 10$ 。所对应的  $\mathbb{P}(M_T^N < \epsilon)$  来调整  $M$ ，若对应的概率在  $10^{-2}$  范围内，则将  $M$  减小为原先的一半，否则增大  $M$  为原先的两倍。重复此过程，知道发现满足条件的最小的  $M$ 。我们然后固定这个最优的  $M$ ，然后尝试找出最优的  $N$ 。

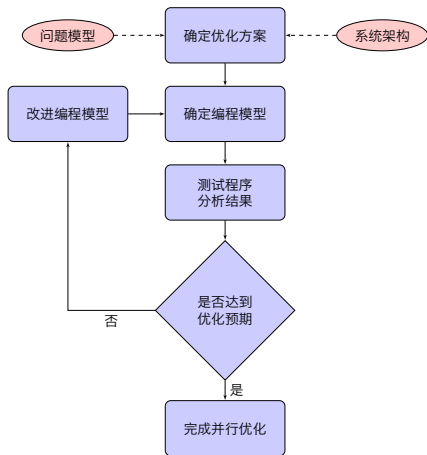
根据前面的理论，最优的  $N$  存在于  $[0, N_0]$  区间内。我们仍然使用二分法寻找最优的  $N$ ，首先我们令  $N_1 = N_0/2$ ，在此参数下模拟相应的  $\mathbb{P}(M_T^N < \epsilon)$ ，若此模拟概率大于  $Prob$ ，则令  $N_2 = N_1/2$ ，反之，则令  $N_2 = \frac{N_0 + N_1}{2}$ 。重复此过程直至找到最优的  $N$ 。



# 内容提要

- 1 背景介绍
- 2 课题陈述
- 3 并行化算法**
- 4 实验结果及分析

# 常用的并行化策略



## 针对内存使用的优化

- 对齐数据(data alignment)
- 数据块(cache blocking)
- 预读取(prefetching)
- 流式存储技术(streaming store)

## 针对 for 循环的改进

- 循环展开(loop unrolling)
- 分块循环(loop tiling)
- 循环互换(loop interchange)
- 循环合并(loop fusion)
- 循环偏移(loop skewing)
- 循环剥离(loop peeling)

# 算法并行性分析

```
procedure BSERROR(M, N)
  error  $\leftarrow$  0
  NRV[N]
  BM[N]
  PX[N + 1]
   $\delta t \leftarrow T/N$ 
  count  $\leftarrow$  0
  for m = 1 : M do
    error  $\leftarrow$  0
    for all NRV[j] do NRV[j]  $\leftarrow$  GaussianNumGenerator()
    end for
    BM[0]  $\leftarrow$  NRV[0]  $\cdot \sqrt{\delta t}$ 
    for all BM[j] do BM[j]  $\leftarrow$  BM[j - 1] + NRV[j]  $\cdot \sqrt{\delta t}$ 
    end for
    PX[0]  $\leftarrow$  X0
    for all j = 1 : N + 1 do
      PX[j]  $\leftarrow$  X0  $\cdot \exp(-0.5 \cdot \sigma^2 \cdot j \cdot \delta t + \sigma \cdot BM[j - 1])$ 
    end for
    for all j = 0 : N - 1 do
      Upper  $\leftarrow$  (log(PX[j]/K) + 0.5  $\cdot \sigma^2 \cdot (T - T_j)) / (\sigma \cdot \sqrt{T - T_j})$ 
      error  $\leftarrow$  error - 1/(\sqrt{2\pi})  $\cdot (PX[j + 1] - PX[j]) \cdot \int_{-\infty}^{Upper} e^{-t^2/2} dt$ 
    end for
  end for
end procedure
```

# 算法并行性分析

**procedure** BSERROR(M, N)

  error  $\leftarrow$  0

  NRV[N]

  BM[N]

  PX[N + 1]

$\delta t \leftarrow T/N$

  count  $\leftarrow$  0

**for**  $m = 1 : M$  **do**

    error  $\leftarrow$  0

**for all** NRV[j] **do** NRV[j]  $\leftarrow$  GaussianNumGenerator()

**end for**

    BM[0]  $\leftarrow$  NRV[0]  $\cdot \sqrt{\delta t}$

**for all** BM[j] **do** BM[j]  $\leftarrow$  BM[j - 1] + NRV[j]  $\cdot \sqrt{\delta t}$

**end for**

    PX[0]  $\leftarrow$  X0

**for all**  $j = 1 : N + 1$  **do**

      PX[j]  $\leftarrow$  X0  $\cdot \exp(-0.5 \cdot \sigma^2 \cdot j \cdot \delta t + \sigma \cdot BM[j - 1])$

**end for**

**for all**  $j = 0 : N - 1$  **do**

      Upper  $\leftarrow (\log(PX[j]/K) + 0.5 \cdot \sigma^2 \cdot (T - T_j)) / (\sigma \cdot \sqrt{T - T_j})$

      error  $\leftarrow$  error -  $1/(\sqrt{2\pi}) \cdot (PX[j + 1] - PX[j]) \cdot \int_{-\infty}^{Upper} e^{-t^2/2} dt$

**end for**

**end for**

**end procedure**

# 算法并行性分析

**procedure** BSERROR(M, N)

  error  $\leftarrow$  0

  NRV[N]

  BM[N]

  PX[N + 1]

$\delta t \leftarrow T/N$

  count  $\leftarrow$  0

**for**  $m = 1 : M$  **do**

    error  $\leftarrow$  0

**for all** NRV[j] **do** NRV[j]  $\leftarrow$  GaussianNumGenerator()

**end for**

    BM[0]  $\leftarrow$  NRV[0]  $\cdot \sqrt{\delta t}$

**for all** BM[j] **do** BM[j]  $\leftarrow$  BM[j - 1] + NRV[j]  $\cdot \sqrt{\delta t}$

**end for**

    PX[0]  $\leftarrow$  X0

**for all** j = 1 : N + 1 **do**

      PX[j]  $\leftarrow$  X0  $\cdot \exp(-0.5 \cdot \sigma^2 \cdot j \cdot \delta t + \sigma \cdot BM[j - 1])$

**end for**

**for all** j = 0 : N - 1 **do**

      Upper  $\leftarrow (\log(PX[j]/K) + 0.5 \cdot \sigma^2 \cdot (T - T_j)) / (\sigma \cdot \sqrt{T - T_j})$

      error  $\leftarrow$  error -  $1/(\sqrt{2\pi}) \cdot (PX[j + 1] - PX[j]) \cdot \int_{-\infty}^{Upper} e^{-t^2/2} dt$

**end for**

**end for**

**end procedure**

# 算法并行性分析

**procedure** BSERROR(M, N)

error  $\leftarrow$  0

NRV[N]

BM[N]

PX[N + 1]

$\delta t \leftarrow T/N$

count  $\leftarrow$  0

**for** m = 1 : M **do**

error  $\leftarrow$  0

**for all** NRV[j] **do** NRV[j]  $\leftarrow$  GaussianNumGenerator()

**end for**

BM[0]  $\leftarrow$  NRV[0]  $\cdot \sqrt{\delta t}$

**for all** BM[j] **do** BM[j]  $\leftarrow$  BM[j - 1] + NRV[j]  $\cdot \sqrt{\delta t}$

**end for**

PX[0]  $\leftarrow$  X0

**for all** j = 1 : N + 1 **do**

PX[j]  $\leftarrow$  X0  $\cdot \exp(-0.5 \cdot \sigma^2 \cdot j \cdot \delta t + \sigma \cdot BM[j - 1])$

**end for**

**for all** j = 0 : N - 1 **do**

Upper  $\leftarrow (\log(PX[j]/K) + 0.5 \cdot \sigma^2 \cdot (T - T_j)) / (\sigma \cdot \sqrt{T - T_j})$

error  $\leftarrow$  error -  $1/(\sqrt{2\pi}) \cdot (PX[j + 1] - PX[j]) \cdot \int_{-\infty}^{Upper} e^{-t^2/2} dt$

**end for**

**end for**

**end procedure**

蓝色的外层循环为相互独立的蒙特卡洛模拟次数并行度高, 可并行在单个 MIC 的不同线程上也可以并行在多个 MIC 处理器上

# 算法并行性分析

**procedure** BSERROR(M, N)

error  $\leftarrow$  0

NRV[N]

BM[N]

PX[N + 1]

$\delta t \leftarrow T/N$

count  $\leftarrow$  0

**for** m = 1 : M **do**

error  $\leftarrow$  0

**for all** NRV[j] **do** NRV[j]  $\leftarrow$  GaussianNumGenerator()

**end for**

BM[0]  $\leftarrow$  NRV[0]  $\cdot \sqrt{\delta t}$

**for all** BM[j] **do** BM[j]  $\leftarrow$  BM[j - 1] + NRV[j]  $\cdot \sqrt{\delta t}$

**end for**

PX[0]  $\leftarrow$  X0

**for all** j = 1 : N + 1 **do**

PX[j]  $\leftarrow$  X0  $\cdot \exp(-0.5 \cdot \sigma^2 \cdot j \cdot \delta t + \sigma \cdot BM[j - 1])$

**end for**

**for all** j = 0 : N - 1 **do**

Upper  $\leftarrow (\log(PX[j]/K) + 0.5 \cdot \sigma^2 \cdot (T - T_j)) / (\sigma \cdot \sqrt{T - T_j})$

error  $\leftarrow$  error -  $1/(\sqrt{2\pi}) \cdot (PX[j + 1] - PX[j]) \cdot \int_{-\infty}^{Upper} e^{-t^2/2} dt$

**end for**

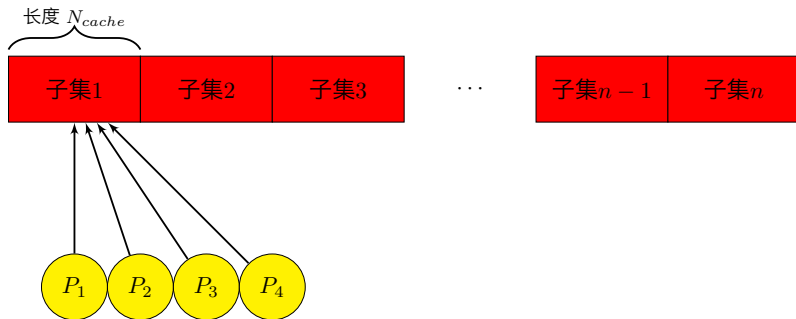
**end for**

**end procedure**

蓝色的外层循环为相互独立的蒙特卡洛模拟次数并行度高, 可并行在单个 MIC 的不同线程上也可以并行在多个 MIC 处理器上

红色循环为离线状态数组, 数组前后状态有依赖关系, 需要进行并行优化

# 单片 MIC 单次蒙特卡洛并行优化

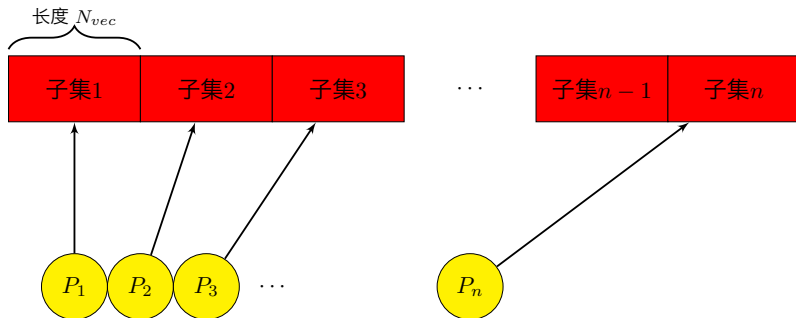


## 方案一

离线状态的数组总长度  $N$  分为  $n$  个子集，每个的长度为  $N_{cache}$ ，单片 MIC 上的所有线程依次同时并行工作于子集 1，子集 2，直至最后一个子集。 $N_{cache}$  的大小要和缓存大小匹配，使数据可以在缓存内被所有线程所共享( $512K \times 61 = 31M$ ，所有线程共享一个随机数发生流)



# 单片 MIC 单次蒙特卡洛并行优化



## 方案二

离线状态的数组总长度  $N$  分为  $n$  个子集，每个的长度为  $N_{vec}$ ，每个子集由一个线程负责，所有线程同时并行工作，每个线程拥有自己的随机数发生流， $N_{vec}$  要和单个核心的  $L2$  缓存相符合(512K)，以达到最大的数据重用。

## 基于汇编的矢量化积分运算

并行算法中大量使用了如下形式的积分运算

$$f(x) = \int_{-\infty}^x e^{-\frac{t^2}{2}} dt \quad (4)$$

利用辛普森积分法(Simpson's rule)可以对高斯积分进行矢量化(vectorized)计算

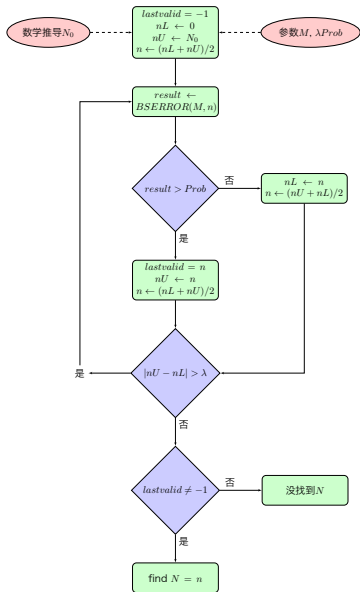
转化为高斯积分

$$f(x) = \int_{-\infty}^x e^{-\frac{t^2}{2}} dt = 0.5 \times \sqrt{2\pi} + \int_0^x e^{-\frac{t^2}{2}} dt \quad (5)$$

Simpson's rule

$$\int_a^b f(x) dx \approx \frac{h}{3} \left[ f(x_0) + 2 \sum_{j=1}^{n/2-1} f(x_{2j}) + 4 \sum_{j=1}^{n/2} f(x_{2j-1}) + f(x_n) \right] \quad (6)$$

# 基于二分法的 $N$ 值最优化搜索



## 初始值 $N_0$

本文通过数学推导给出了一个  $N$  值的上界  $N_0$ , 从而使我们的二分法有了初始的搜索上界

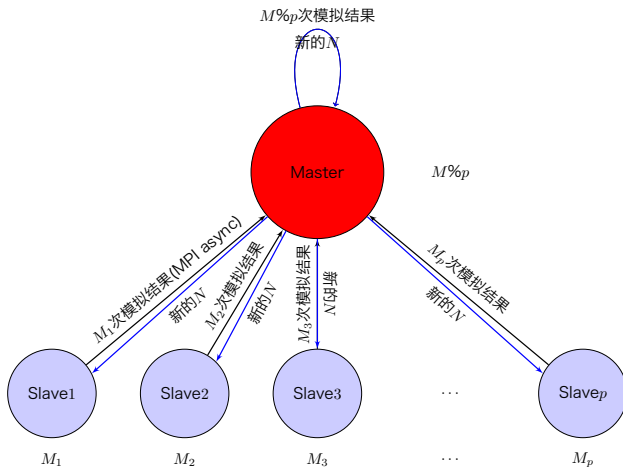
## 参数 $\lambda, Prob, M$

$\lambda$  是设置的最小区间, 当上下界之差小于  $\lambda$  时, 认为二分法结束  $Prob$  则为置信概率, 值越高表明最后搜索到得  $N$  值可信度越高。  $M$  为蒙特卡洛模拟的次数, 越大的  $M$  也说明搜索到的结果越可靠。

# 多 MIC 并行优化

## Maser-Slave 模式

一个节点被选为 Master，其余节点为 Slave 节点在  $M$  的维度上进行并行优化。

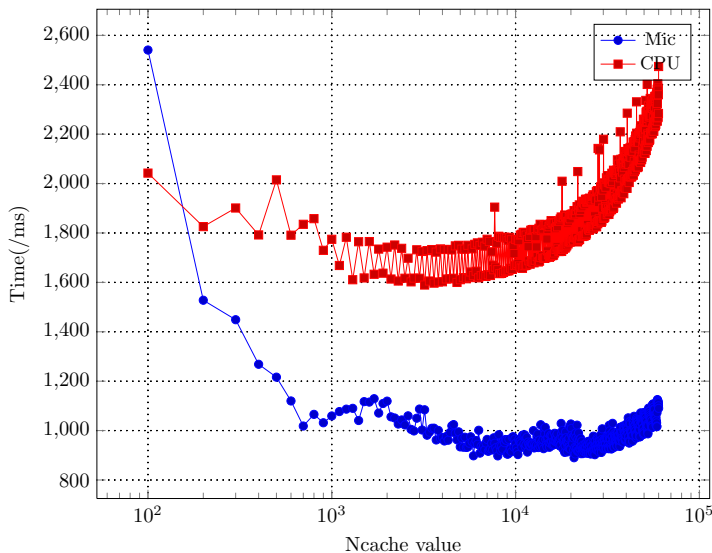


# 内容提要

- 1 背景介绍
- 2 课题陈述
- 3 并行化算法
- 4 实验结果及分析**

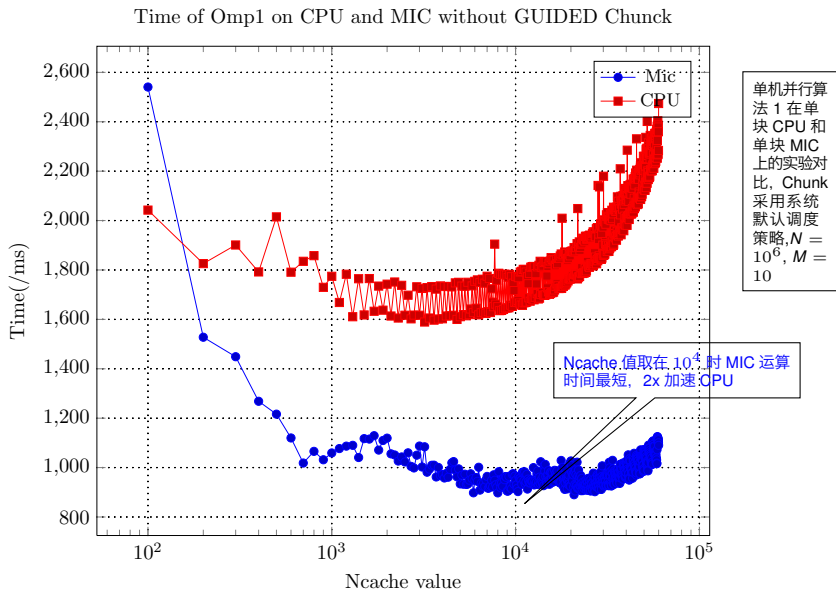
# 单 MIC 及 CPU 版本的实验对比(1/2)

Time of Omp1 on CPU and MIC without GUIDED Chunk

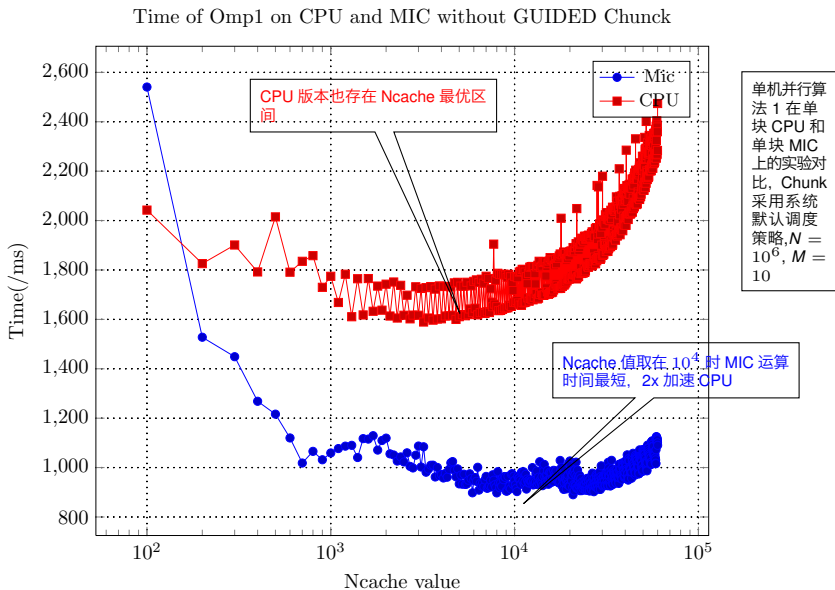


单机并行算法 1 在单块 CPU 和单块 MIC 上的实验对比, Chunk 采用系统默认调度策略,  $N = 10^6$ ,  $M = 10$

# 单 MIC 及 CPU 版本的实验对比(1/2)



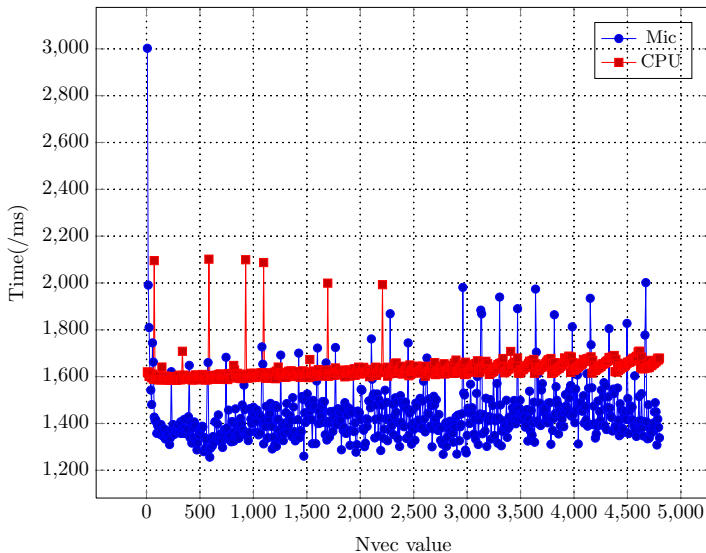
# 单 MIC 及 CPU 版本的实验对比(1/2)





# 单 MIC 及 CPU 版本的实验对比(2/2)

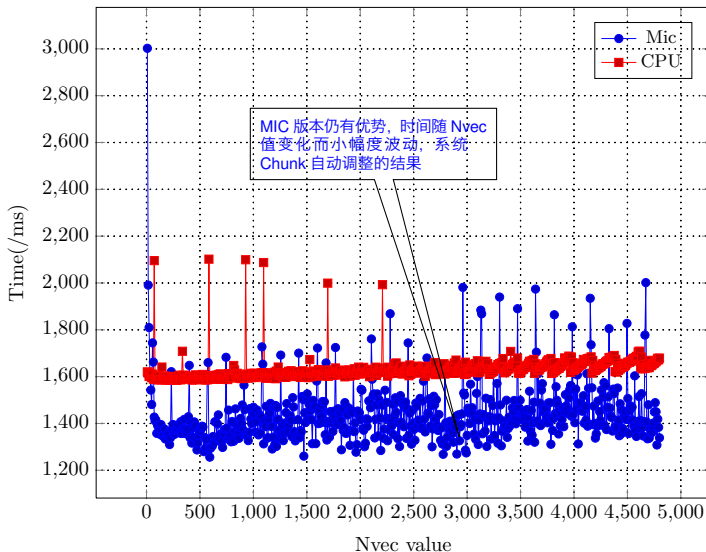
Time of Omp2 on CPU and MIC without GUIDED Chunk



单机并行算法 2 在单块 CPU 和单块 MIC 上的实验对比, Chunk 采用系统默认调度策略,  $N = 10^6$ ,  $M = 10$

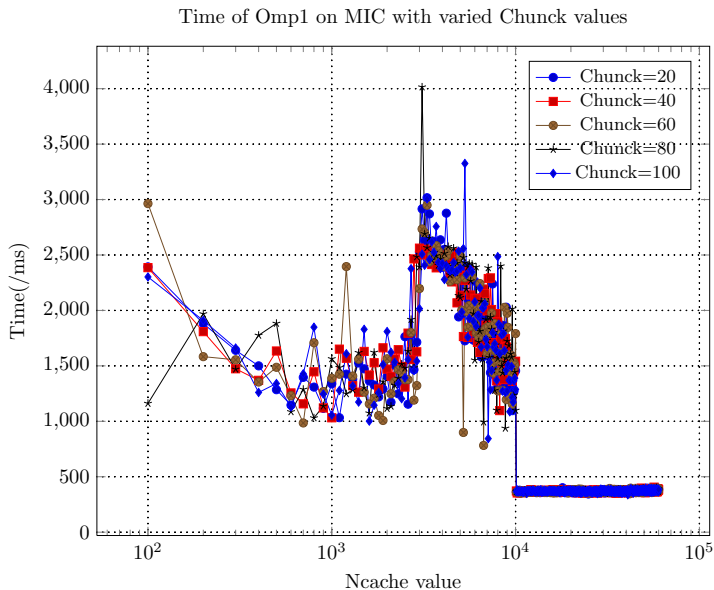
## 单 MIC 及 CPU 版本的实验对比(2/2)

Time of Omp2 on CPU and MIC without GUIDED Chunk



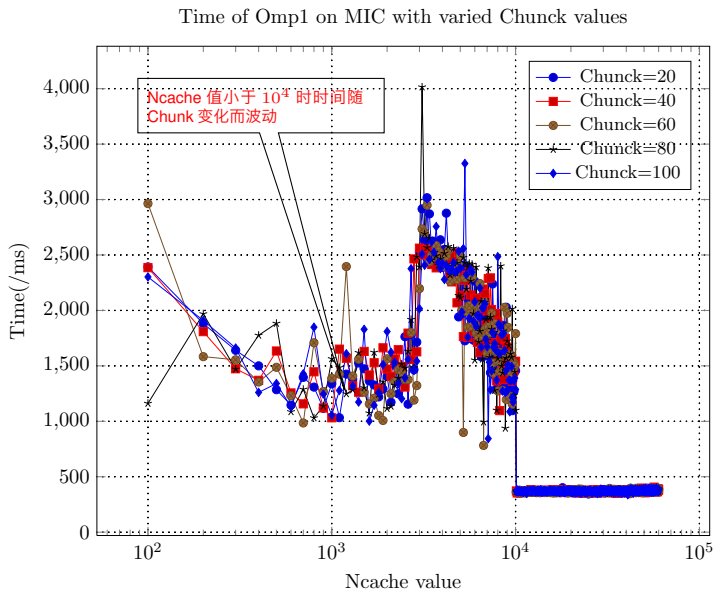
单机并行算法 2 在单块 CPU 和单块 MIC 上的实验对比, Chunk 采用系统默认调度策略,  $N = 10^6$ ,  $M = 10$

# Chunk 取值对算法 1 的影响

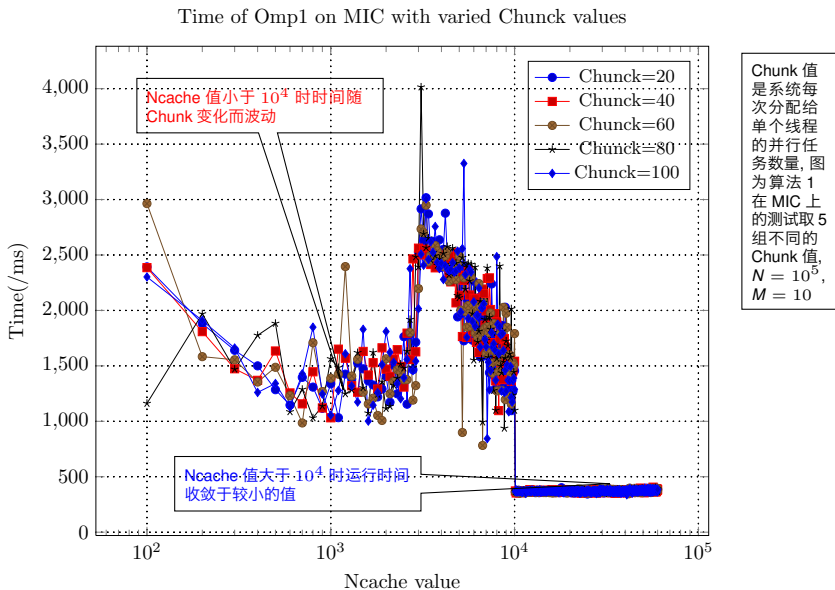


Chunk 值是系统每次分配给单个线程的并行任务数量, 图为算法 1 在 MIC 上的测试取 5 组不同的 Chunk 值,  $N = 10^5$ ,  $M = 10$

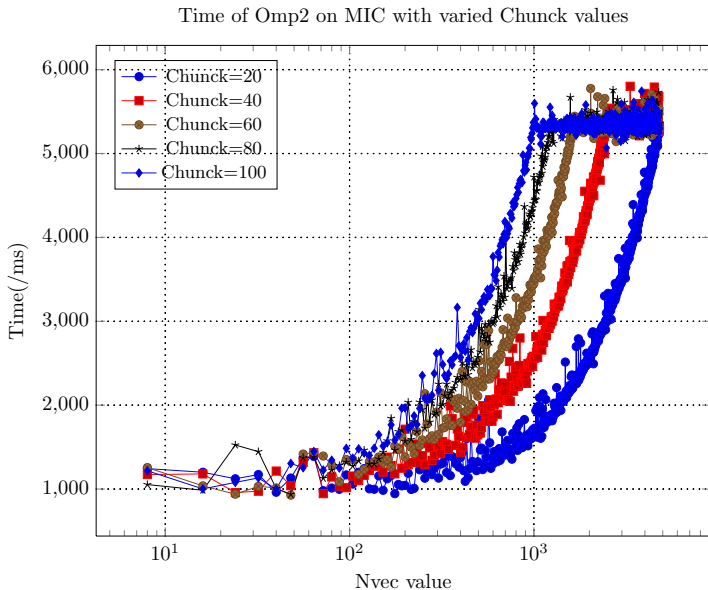
# Chunk 取值对算法 1 的影响



# Chunk 取值对算法 1 的影响



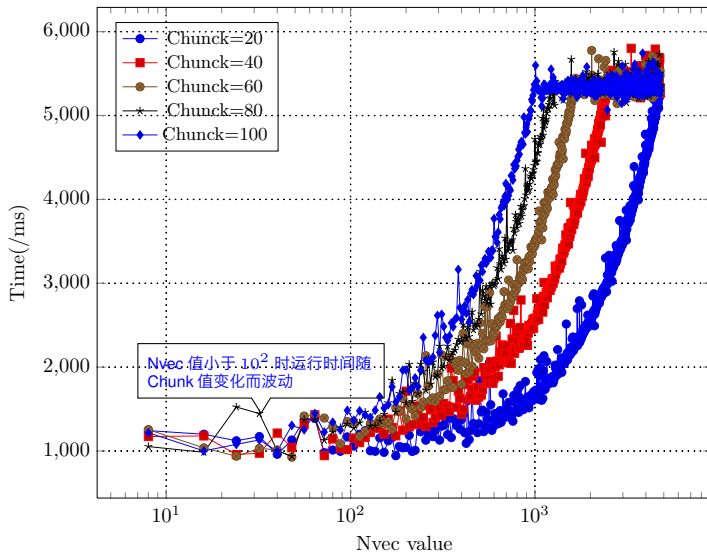
# Chunk 取值对算法 2 的影响



Chunk 值是系统每次分配给单个线程的并行任务数量, 图为算法 2 在 MIC 上的测试取 5 组不同的 Chunk 值,  $N = 10^5$ ,  $M = 10$

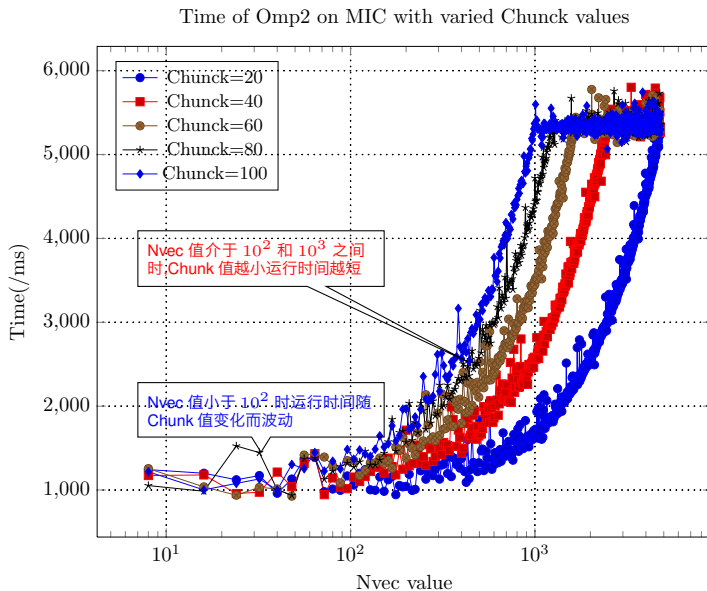
# Chunk 取值对算法 2 的影响

Time of Omp2 on MIC with varied Chunck values



Chunk 值是系统每次分配给单个线程的并行任务数量, 图为算法 2 在 MIC 上的测试取 5 组不同的 Chunk 值,  $N = 10^5$ ,  $M = 10$

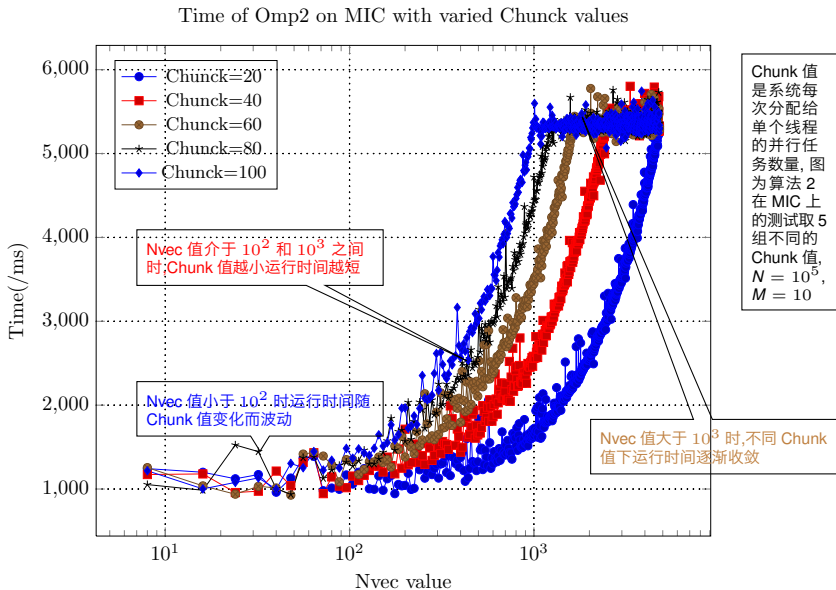
# Chunk 取值对算法 2 的影响



Chunk 值是系统每次分配给单个线程的并行任务数量, 图为算法 2 在 MIC 上的测试取 5 组不同的 Chunk 值,  $N = 10^5$ ,  $M = 10$

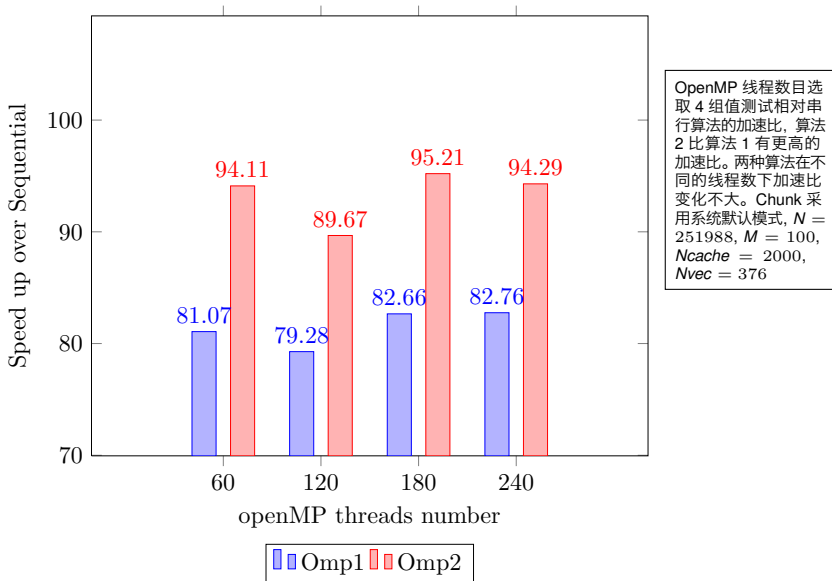


# Chunk 取值对算法 2 的影响

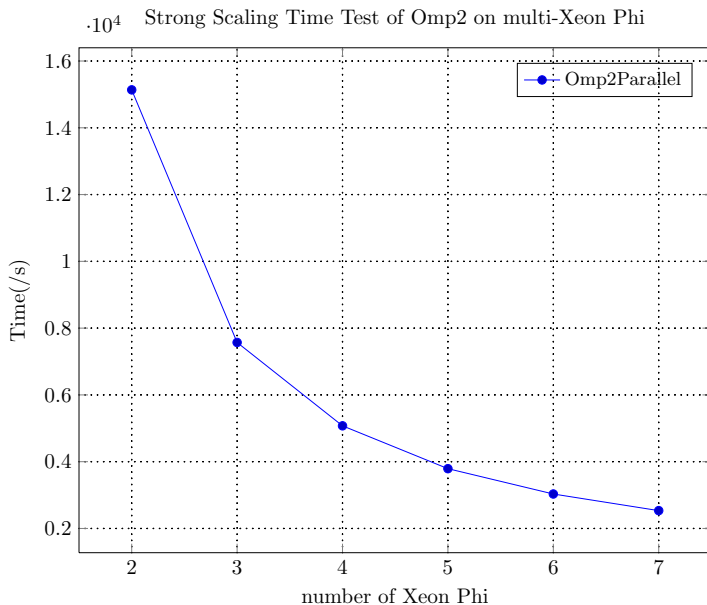


# Omp1 与 Omp2 两种算法的加速对比

Omp1 and Omp2 time on MIC with different number of threads

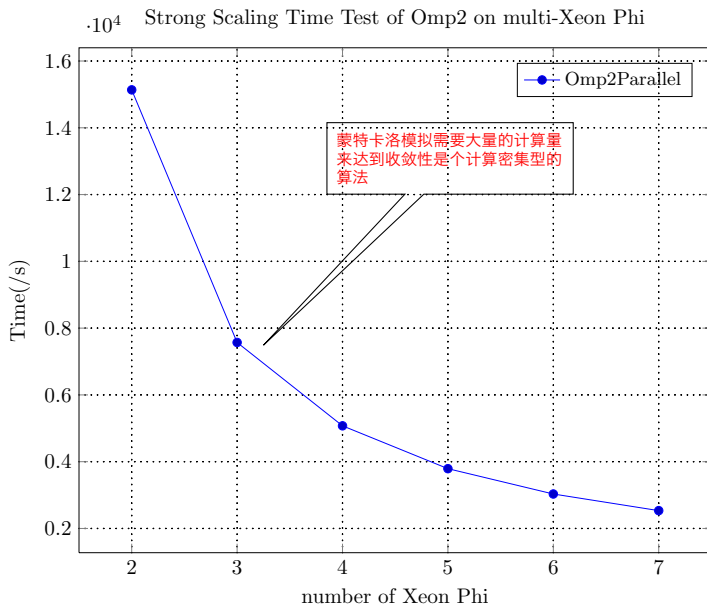


# 多 MIC 优化可扩展性



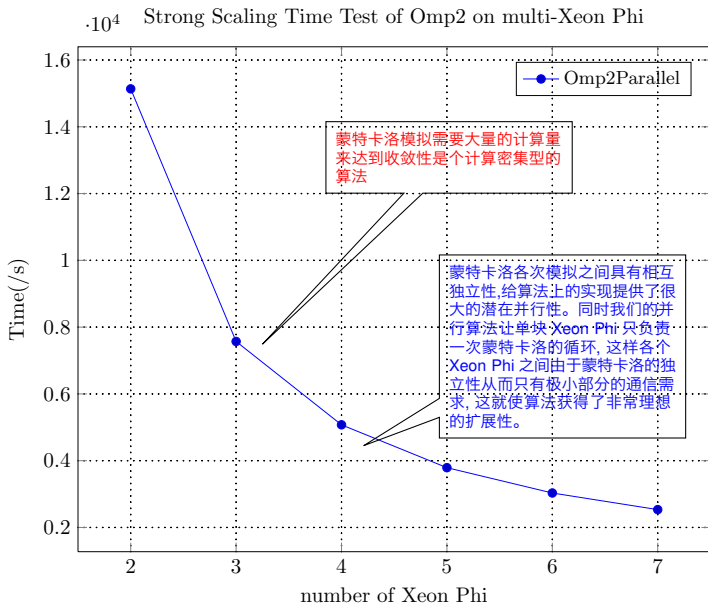
$N = 251988$ ,  
 $M = 10000$ ,  
 $N_{cache} = 2000$ ,  
 $N_{vec} = 192$ 。图中显示我们的算法获得了超线性的扩展性能

# 多 MIC 优化可扩展性



$N = 251988$ ,  
 $M = 10000$ ,  
 $N_{cache} = 2000$ ,  
 $N_{vec} = 192$ 。图中显示我们的算法获得了超线性的扩展性能

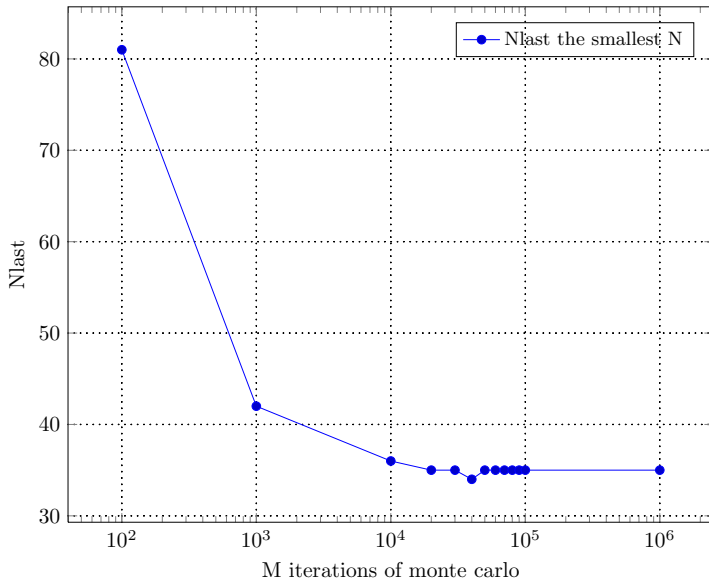
# 多 MIC 优化可扩展性



$N = 251988$ ,  
 $M = 10000$ ,  
 $N_{\text{cache}} = 2000$ ,  
 $N_{\text{vec}} = 192$ 。图中显示我们的算法获得了超线性的扩展性能

# 算法的收敛性研究

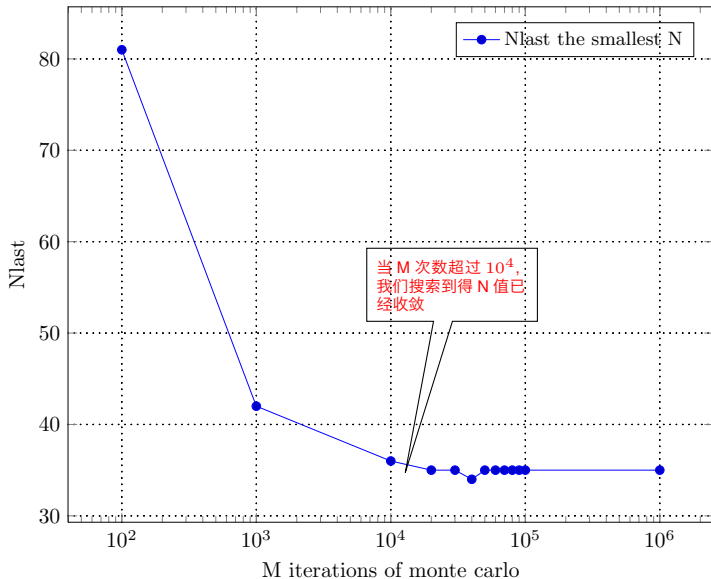
Omp2 convergency test with  $M$



算法 2 在  
多 CPU  
上的收  
敛性,  $N =$   
251988,  
 $N_{vec} =$   
192

# 算法的收敛性研究

Omp2 convergency test with  $M$



算法 2 在多 CPU 上的收敛性,  $N = 251988$ ,  $N_{vec} = 192$

# 结论

- 我们研究了基于 Black-Scholes 模型的离散时间版本的 Delta 对冲策略与理论上的 Delta 对冲策略之间的误差。
- 我们利用高性能计算的相关技术针对 Intel 平台实现了该应用的并行化。
- 我们首先针对单机利用多线程和矢量化提出了两种并行方案，然后使用 Master-Slave 模型利用消息传递模式通过 MPI 实现了多机上运行的并行版本。
- 我们开创性地通过理论给出最优抽样次数的数学上界,然后通过二分法在由该上界给出的闭区间内搜索最优解。
- 测试表明并行程序相对于串行版本获得了极大的性能提升。并行程序在 MIC 集群上也获得了良好的扩展性。