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[1,2] 陈浪石[1,3] 潘慈辉[1,4]

[1]Maison de la Simulation

[2]Atomic Energy and Alternative Energies Commission (C.E.A)

[3]French National Centre for Scientific Research (C.N.R.S)

[4]Versailles Saint-Quentin-en-Yvelines University

November 3, 2014

内容提要

内容提要

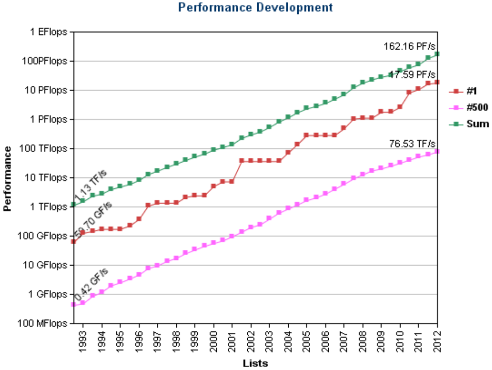
超算发展及 Xeon Phi

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

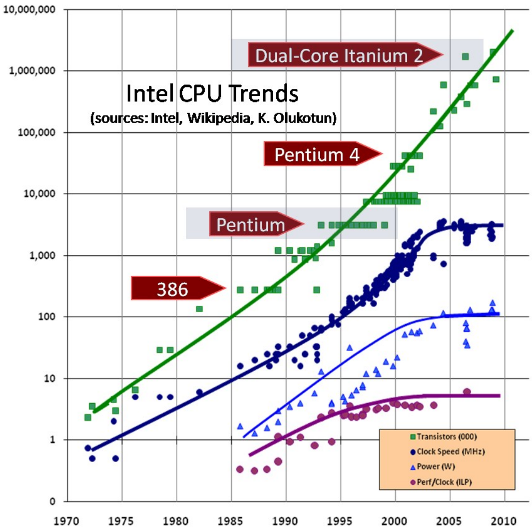
超算发展及 Xeon Phi

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)

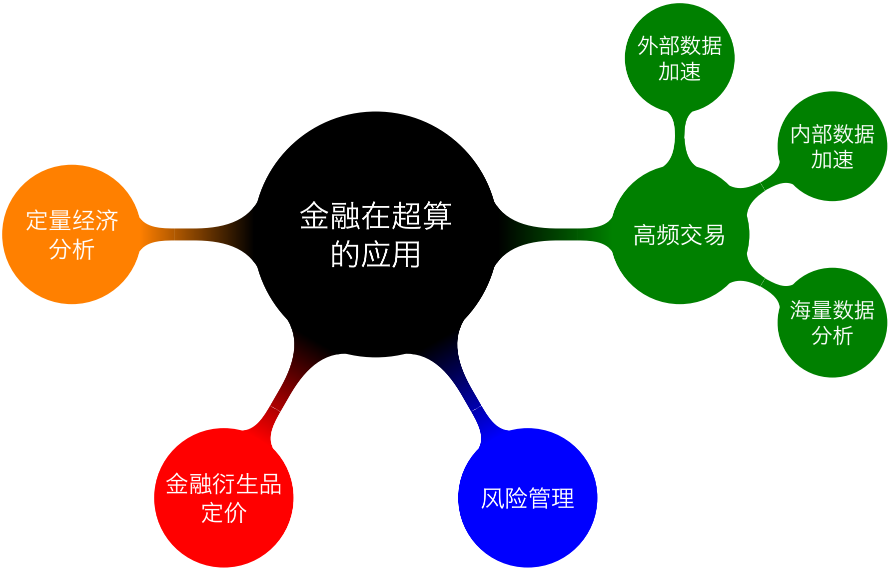
目前众核结构处理器的代表

- 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

欧式期权

content

Black-Scholes 模型

content

对冲策略及误差分析

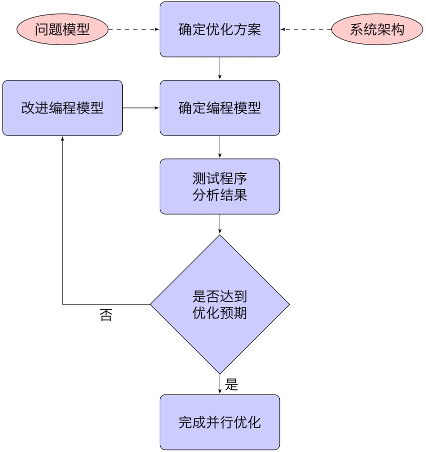
content

参数选择及收敛条件

content

内容提要

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 ← 0
    NRV[N]
    BM[N]
    PX[N + 1]
    δt ← T/N
    count ← 0
    for m = 1 : M do
        error ← 0
        for all NRV[j] do NRV[j] ← GaussianNumGenerator()
        end for
        BM[0] ← NRV[0] · √δt
        for all BM[j] do BM[j] ← BM[j − 1] + NRV[j] · √δt
        end for
        PX[0] ← X0
        for all j = 1 : N + 1 do
            PX[j] ← X0 · exp(−0.5 · σ^2 · j · δt + σ · BM[j − 1])
        end for
        for all j = 0 : N − 1 do
            Upper ← (log(PX[j]/K) + 0.5 · σ^2 · (T − T_j))/(σ · √(T − T_j))
            error ← error − 1/(√(2π)) · (PX[j + 1] − PX[j]) · ∫_{−∞}^{Upper} e^{−t^2/2} dt
        end for
    end for
end procedure
```

算法并行性分析

```
procedure BSERROR(M, N)
    error ← 0
    NRV[N]
    BM[N]
    PX[N + 1]
    δt ← T/N
    count ← 0
    for m = 1 : M do
        error ← 0
        for all NRV[j] do NRV[j] ← GaussianNumGenerator()
        end for
        BM[0] ← NRV[0] · √δt
        for all BM[j] do BM[j] ← BM[j − 1] + NRV[j] · √δt
        end for
        PX[0] ← X0
        for all j = 1 : N + 1 do
            PX[j] ← X0 · exp(−0.5 · σ^2 · j · δt + σ · BM[j − 1])
        end for
        for all j = 0 : N − 1 do
            Upper ← (log(PX[j]/K) + 0.5 · σ^2 · (T − T_j))/(σ · √(T − T_j))
            error ← error − 1/(√(2π)) · (PX[j + 1] − PX[j]) · ∫_{−∞}^{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

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

```
procedure BSERROR(M, N)
    error ← 0
    NRV[N]
    BM[N]
    PX[N + 1]
    δt ← T/N
    count ← 0
    for m = 1 : M do
        error ← 0
        for all NRV[j] do NRV[j] ← GaussianNumGenerator()
        end for
        BM[0] ← NRV[0] · √δt
        for all BM[j] do BM[j] ← BM[j − 1] + NRV[j] · √δt
        end for
        PX[0] ← X0
        for all j = 1 : N + 1 do
            PX[j] ← X0 · exp(−0.5 · σ^2 · j · δt + σ · BM[j − 1])
        end for
        for all j = 0 : N − 1 do
            Upper ← (log(PX[j]/K) + 0.5 · σ^2 · (T − T_j))/(σ · √(T − T_j))
            error ← error − 1/(√(2π)) · (PX[j + 1] − PX[j]) · ∫_{−∞}^{Upper} e^{−t^2/2} dt
        end for
    end for
end procedure
```

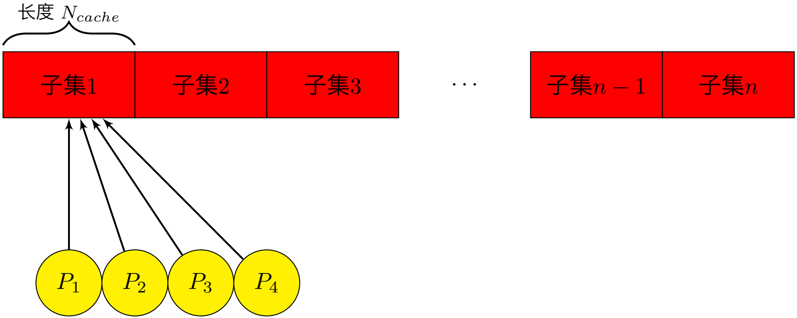
算法并行性分析

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

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

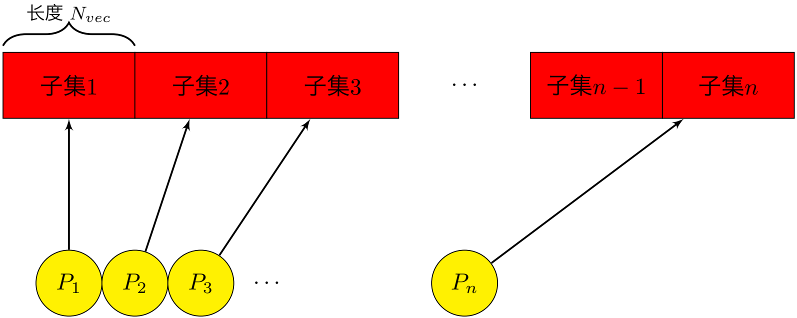
单片 MIC 单次蒙特卡洛并行优化



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

$$f(x) = \int_{-\infty}^{x} e^{-\frac{t^2}{2}} dt \quad (1)$$

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

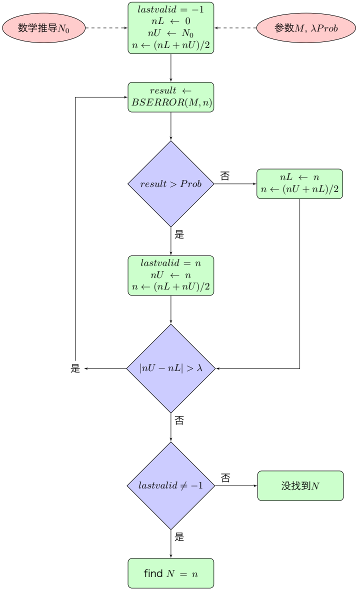
转化为高斯积分

$$f(x) = \int_{-\infty}^{x} e^{-\frac{t^2}{2}} dt = 0.5 \times \sqrt{2\pi} + \int_{0}^{x} e^{-\frac{t^2}{2}} dt \quad (2)$$

Simpson’s rule

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

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

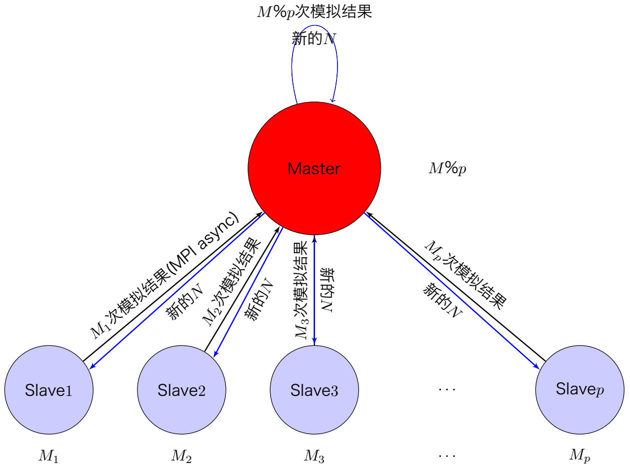
参数 λ, *Prob*, *M*

λ 是设置的最小区间，当上下界之差小于 λ 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

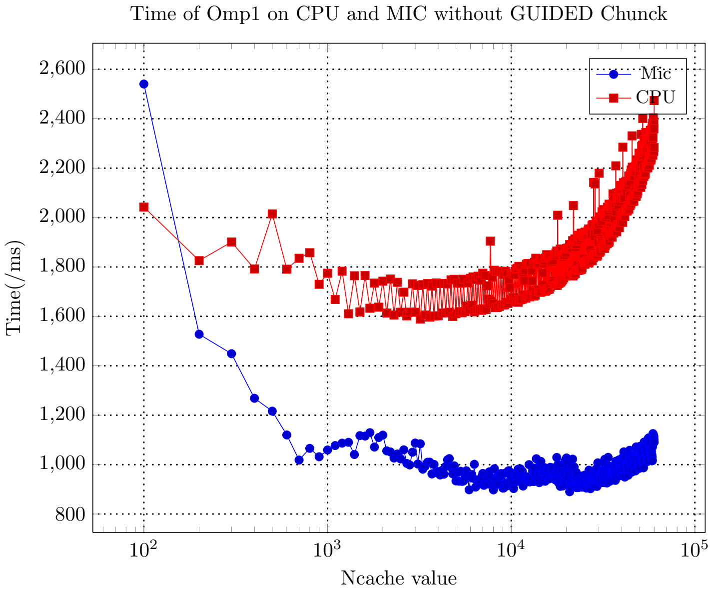
Maser-Slave 模式

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。



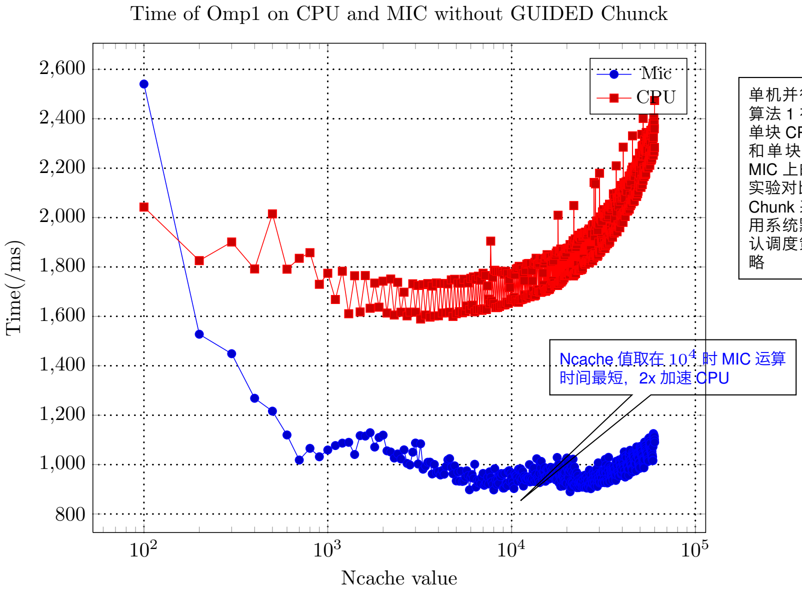
内容提要

单 MIC 及 CPU 版本的实验对比(1/2)



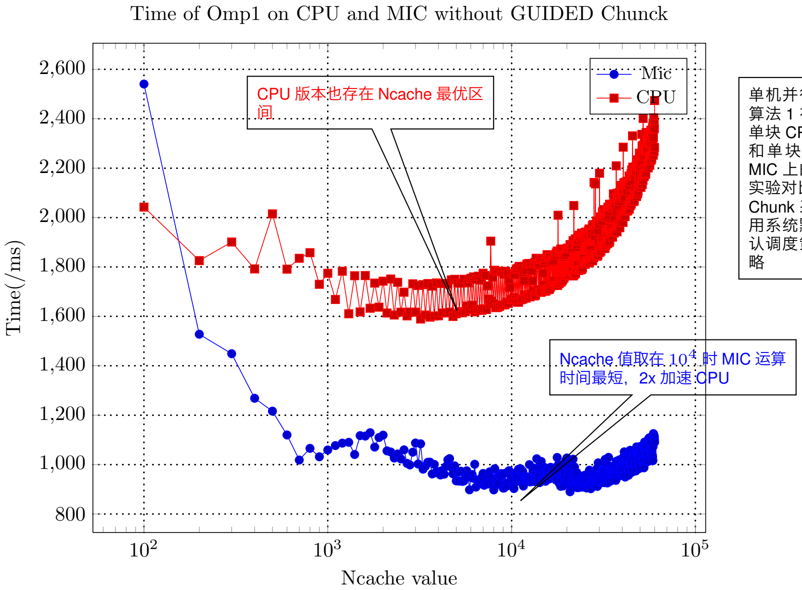
单机并行算法 1 在单块 CPU 和单块 MIC 上的实验对比，Chunk 采用系统默认调度策略

单 MIC 及 CPU 版本的实验对比(1/2)



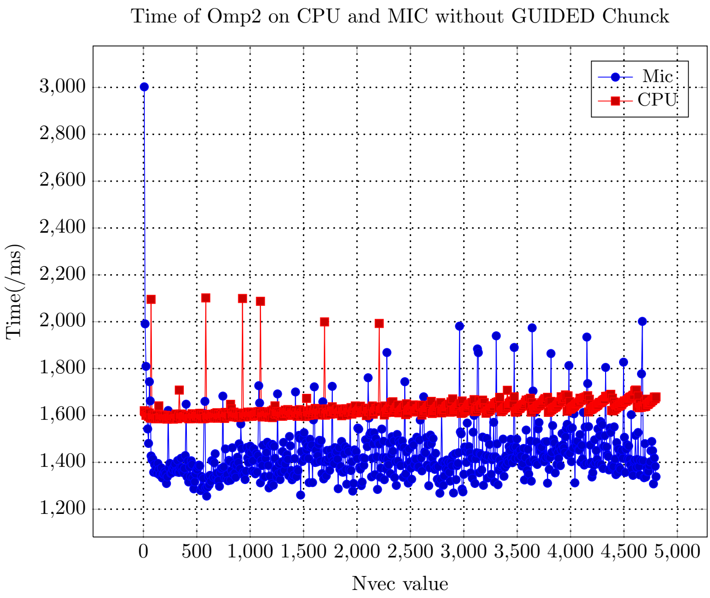
单机并行算法 1 在单块 CPU 和单块 MIC 上的实验对比，Chunk 采用系统默认调度策略

单 MIC 及 CPU 版本的实验对比(1/2)



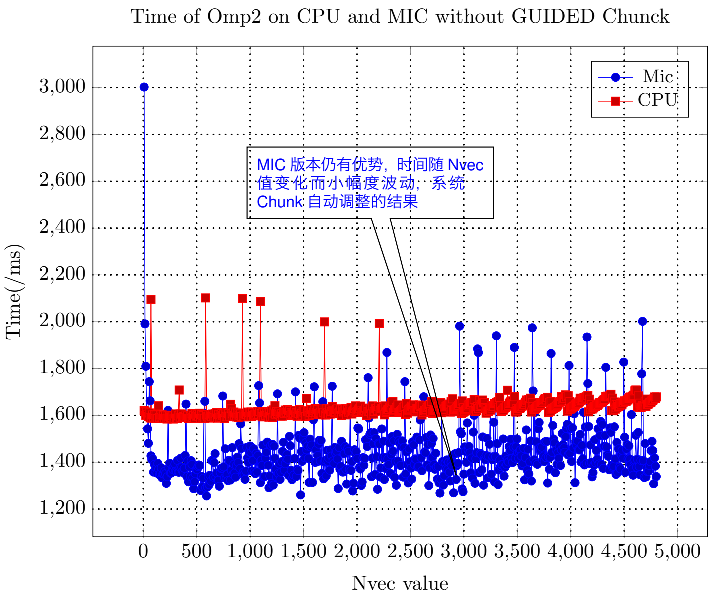
单机并行算法 1 在单块 CPU 和单块 MIC 上的实验对比，Chunk 采用系统默认调度策略

单 MIC 及 CPU 版本的实验对比(2/2)



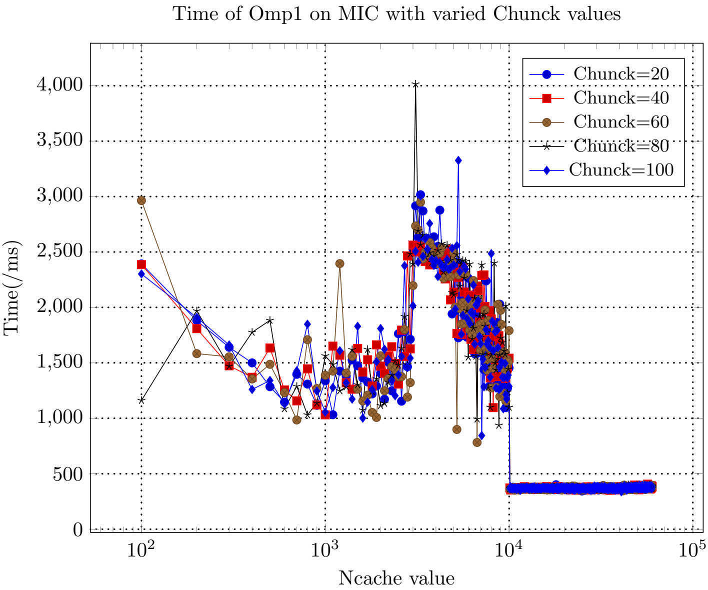
单机并行算法 2 在单块 CPU 和单块 MIC 上的实验对比，Chunk 采用系统默认调度策略

单 MIC 及 CPU 版本的实验对比(2/2)



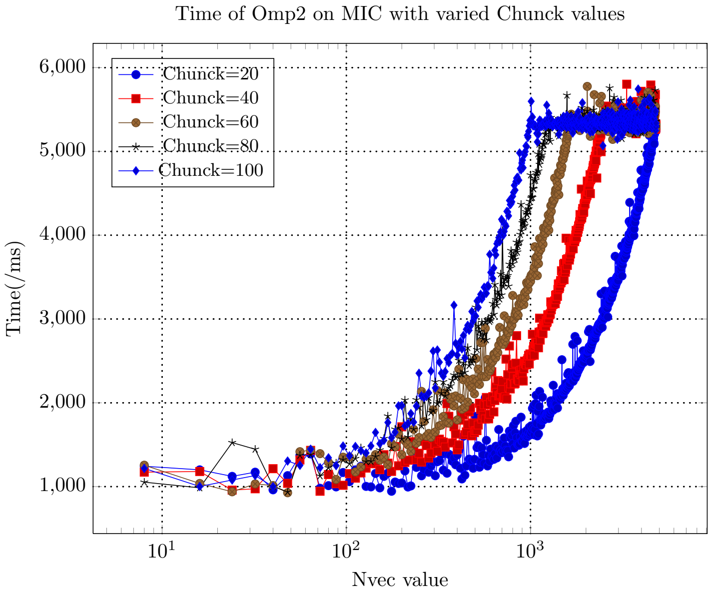
单机并行算法 2 在单块 CPU 和单块 MIC 上的实验对比，Chunk 采用系统默认调度策略

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

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

多 MIC 优化实验及结果分析

content

算法的收敛性研究

content

参考文献