
intel
Leap ahead™
Intel Developer
FORUM

针对多核高性能计算集群的混合并行性能优化

Wanqing He
SSG CRT\HPC 推进经理

SSGS004

Intel Developer FORUM

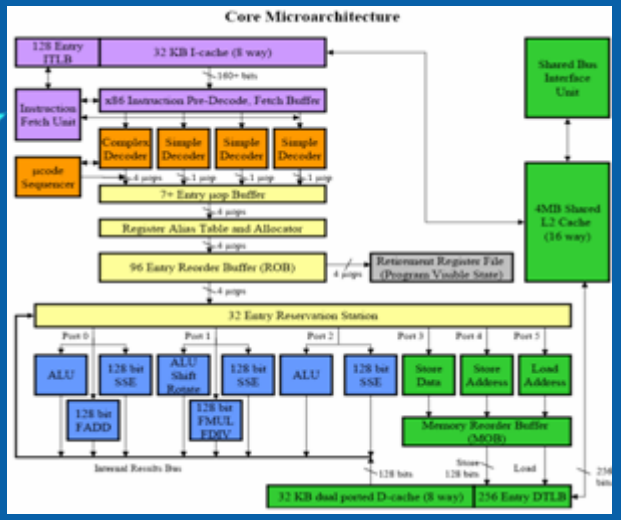
免责声明

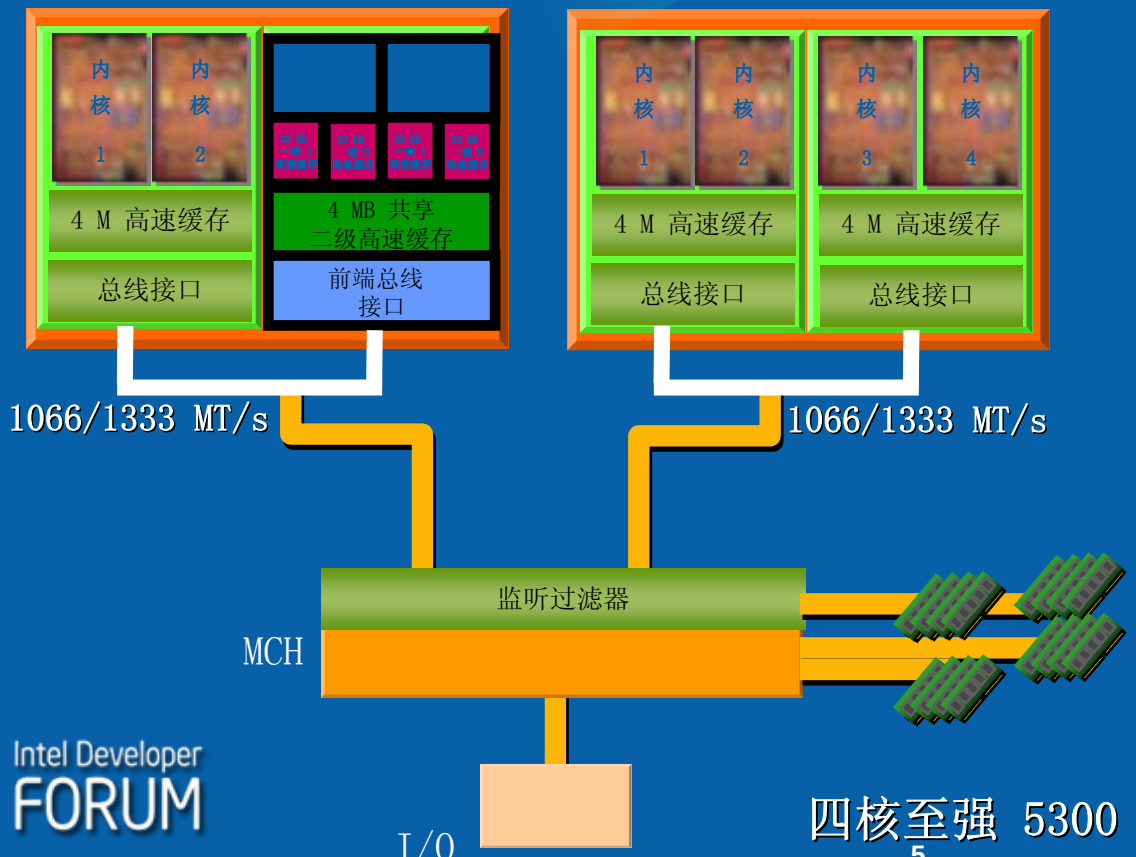
- 本文所提供之信息均与英特尔® 产品相关。除相关产品的英特尔销售条款与条件中列明之担保条件以外，英特尔公司不对销售和/或使用英特尔产品做出任何其它明确或隐含的担保，包括对适用于特定用途、适销性、或不侵犯任何专利、版权或其它知识产权的担保。
- 英特尔可以随时在不发布声明的情况下修改规格、产品说明和计划。
- 提供的日期可能随时更改，恕不另行通知。
- 英特尔、Itanium 和安腾是英特尔公司在美国和其他国家（地区）的商标。
- *文中涉及的其它名称及商标属于各自所有者资产。
-

议程

- 定义混合并行性能调试模型
- 对 HPL 进行性能调试：通过 ITAC 追踪线程抖动
- MPI 和 OpenMP 调试：对真实CFD应用的混合并行
- 尝试使用 Cluster-OpenMP 快速部署集群应用
- 总结/行动号召

英特尔至强5300将酷睿的高性能功耗比引入 SMP 集群



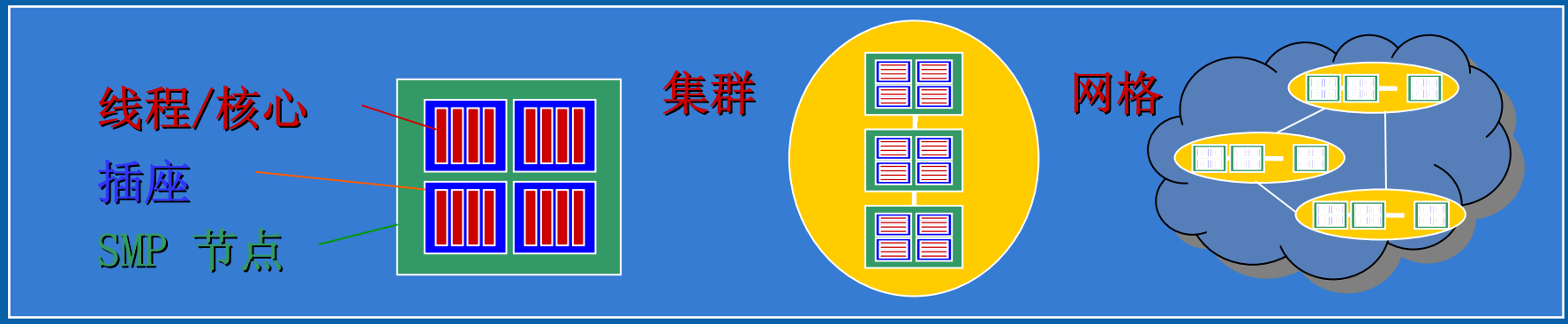


混合并行调试：指 SMP 节点间的 MPI 调试，与 SMP 节点内部的多线程调试的组合，用以同时实现分布式存储并行处理能力和共享存储并行处理能力。

Intel Developer FORUM

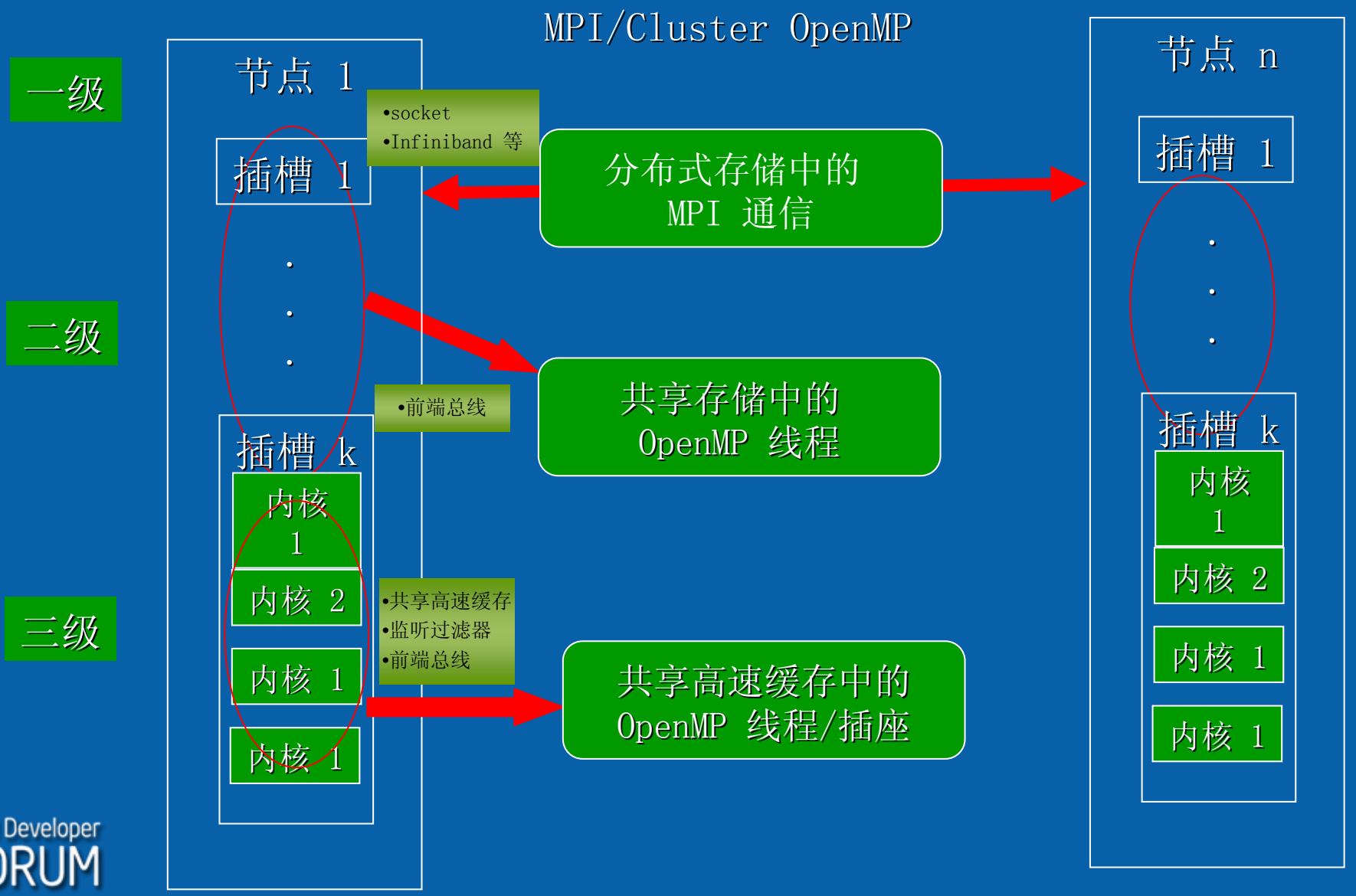
四核至强 5300

高性能计算（HPC）的多核构建模块



HPC 系统	共享存储/节点	分布式存储 /混合集群	计算网格 /
程序模型	在共享存储中采用多线程处理：OpenMP、pthread、winthread	在分布式存储中采用 MPI + OpenMP：英特尔 MPI、ClusterOpenMP	服务导向型架构（SOA）、虚拟化
开发工具	*英特尔® 线程检测器* *英特尔® 线程调节器* *VTune®* *英特尔® 数学核心函数库（MKL）*	*英特尔® MPI* *英特尔® 跟踪收集器/跟踪分析器* *英特尔® 集群数学核心函数库（MKL）* *英特尔® 消息检查器* *英特尔® 线程处理工具*	*UNICORE、Globus* 等等* *DRMAA** *网格编程环境（GPE）*

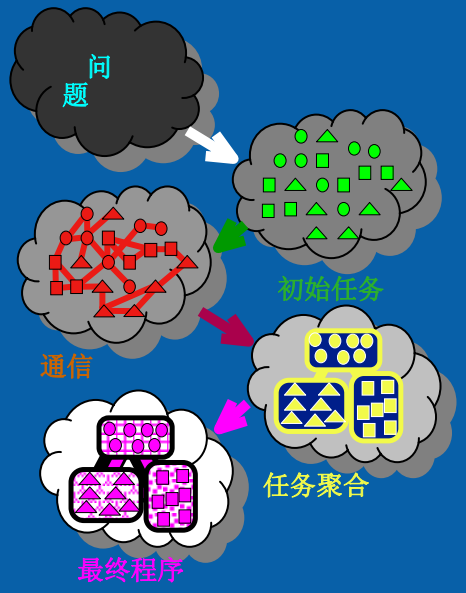
多核 SMP 集群模型



Intel Developer FORUM

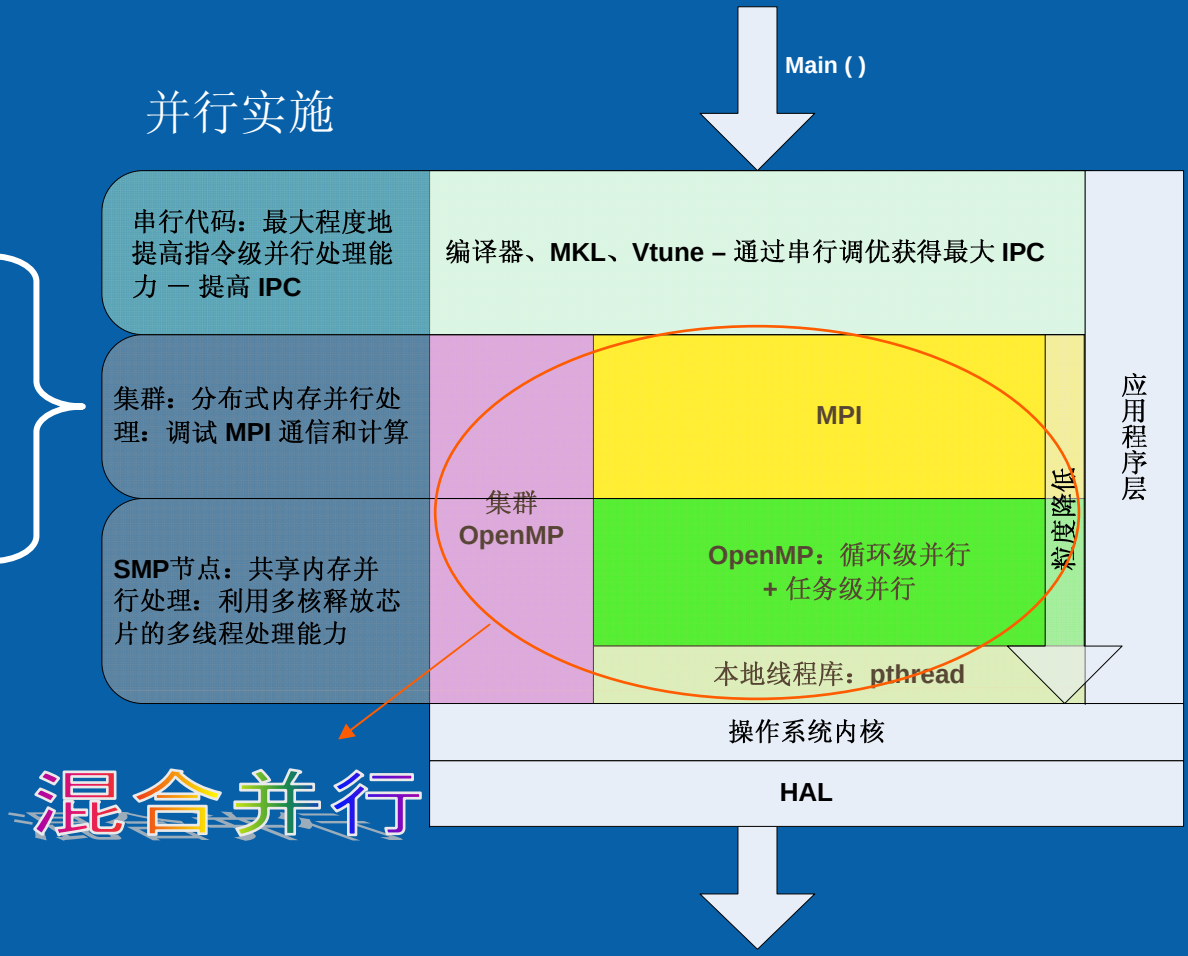
SMP 集群并行模型

混合并行能力应在分布式存储（MPI）、共享存储（OpenMP/线程处理）和指令级并行（串行代码，实现 IPC 提升），从粗粒度到细粒度实施混合并行。



并行设计

并行实施



混合并行

SMP 集群性能模型

- Amdahl 法则的分支：

假定 T1 为串行模式中的执行时间；如果 p 是并行代码的一部分，那么：

$$T_N = (1-p) * T_1 + (p/N) * T_1$$

$$\text{加速} = \frac{1.0}{(1-p) + (p/N)}$$

- 随着我们扩大问题以获得更大的 N，则串行工作量将保持固定，而可并行处理的工作量将随 N 的增加而增加：

$$p \rightarrow 1, \quad Speedup \rightarrow N$$

- *如果我们同时加大问题的规模，则可以实现线性加速（Gustafson 定理）*
- 在混合模型中，$N \approx N_{MPI} * N_{thread} * TR\%$，其中 N_{MPI} 是 MPI 进程的数量，N_{thread} 是一个 MPI 进程中运行的并行线程的数量，TR% 是线程处理比例（代表了一个进程中多线程处理工作的百分比）

真实集群性能模型：MPI

- 考虑到 MPI 的开销时间，我们有：

$$T_N = (1-p)*T_1 + (p/N)*T_1 + T_{mpi}$$

如果考虑到 MPI 模型，通信效果会限制绝对性能和可扩展性：

  - 延迟或消息启动时间（T_L）
  - 带宽，传输开始时的传输速率测量值（B_w）
  - 群集通信调用数
  - 同步，负载失衡

假定 M = 消息平均大小

- T_c =群集通信时间
- T_i = 负载失衡或同步时间

$$T_{mpi} = A1*T_L + A2*M/B_w + A3*T_c + T_i$$

真实集群性能模型：混合模式

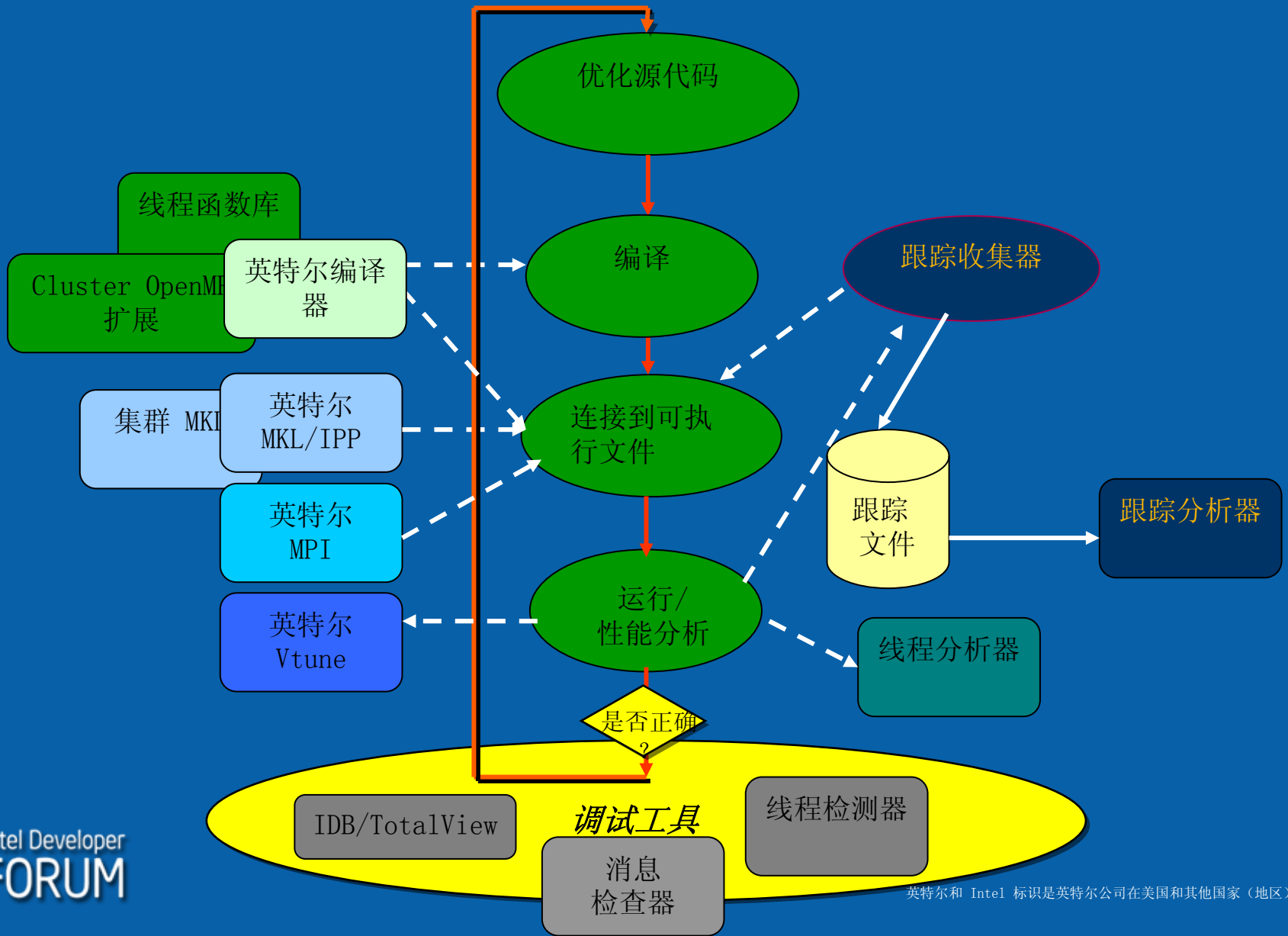
- 如果将多线程处理代码集成到 MPI 模型中，则我们得到

  $T_N = (1-p)*T_1 + (p/(N'_{MPI} * N_{线程} *TR\%))*T_1 + T'_{pi} + \Sigma T_{th}$

  MPI 通信时间变为 $T'_{mpi,}$ 因为存在任务分解的差异，通常 $T'_{mpi} < T_{mpi}$。并行工作扩大到接近 $N'_{MPI} * N_{线程} *TR\%$ 个分割-合并（fork-join）作业数。其中 N'_{MPI} 是新的 MPI 进程数。ΣT_{th} 表示线程在系统中的所有开销，其中可能包括但不限于： 线程生命周期开销、前端总线延迟/带宽开销、共享高速缓存相关开销、监听过滤器开销、线程同步时间，等等。

- 为简单起见，如果 $T'_{mpi} + \Sigma T_{th} < T_{mpi,}$ 则我们可通过将多线程处理引入 SMP 节点来获得整体的性能优势。否则，我们必须进一步调试 OpenMP 来获得更高的并行能力

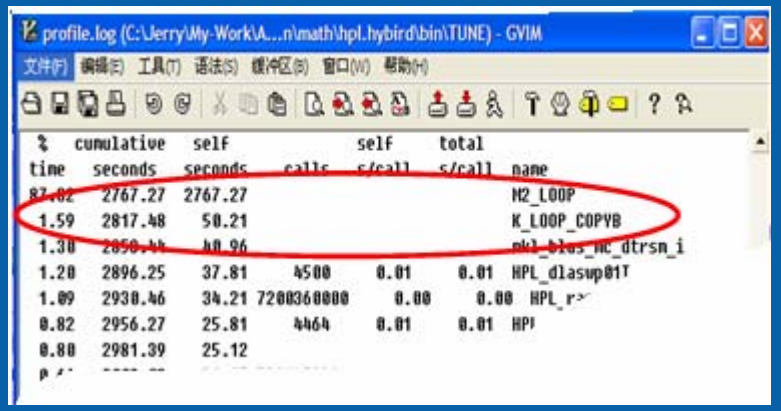
混合编程工具： 编译器/ITAC/ITPT

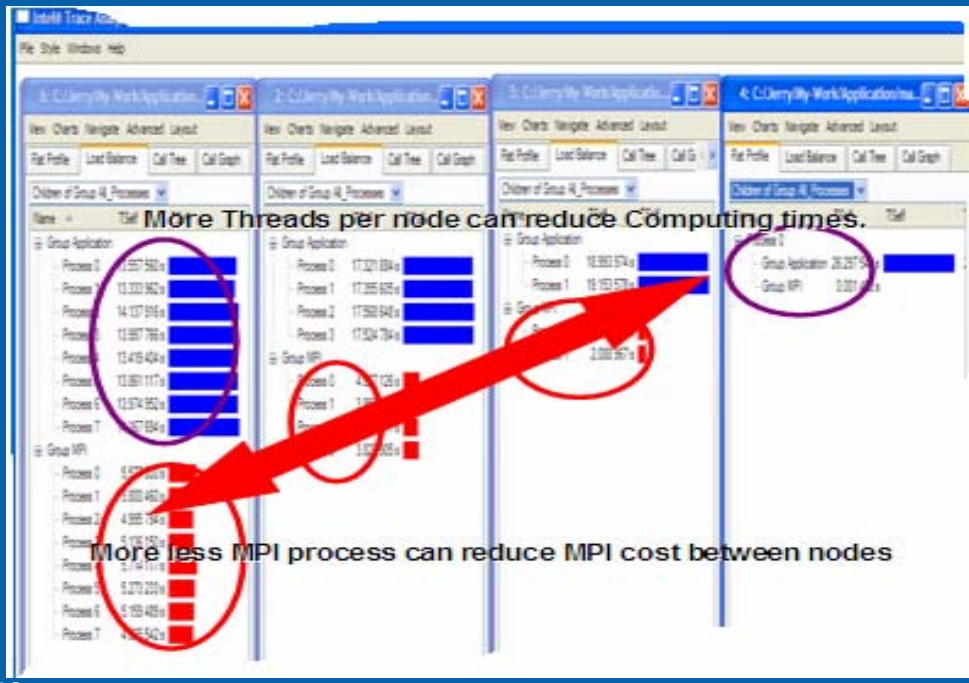


将混合模式应用于 HPL

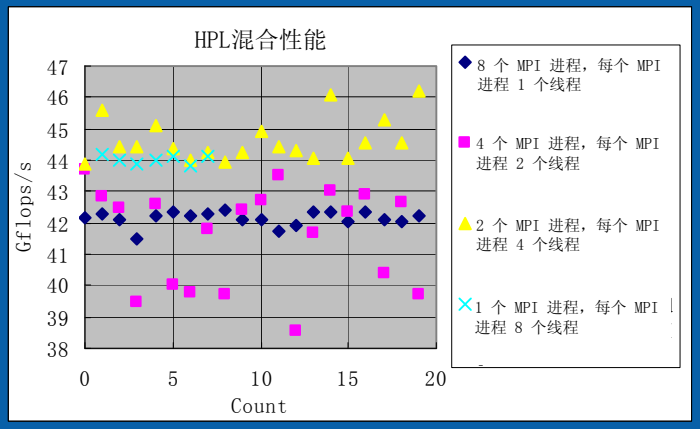
- 在比较单纯 MPI、单纯 OpenMP 和 MPI+OpenMP 版本时发现，HPL能够直接从混合模型中获得的增益并不显著。
- SMP 节点中的 OpenMP 可以切实降低 MPI 通信成本，但对整体性能的影响如何呢？

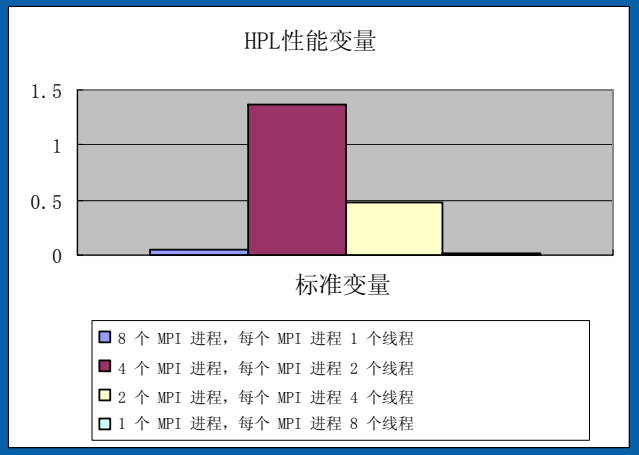
	内核数量（MPI 进程 x 线程）	性能（Gflops/s）
MPI 性能	8 x 1	42.48
OpenMP 性能	1 x 8	44.21
混合性能	2 x 4	46.43



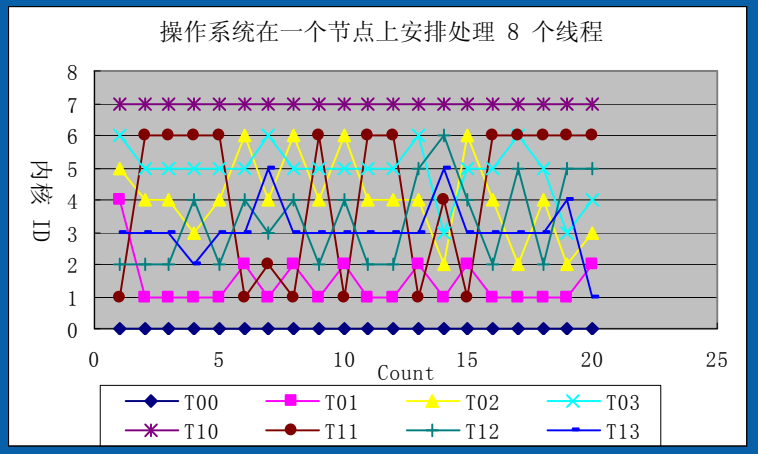


由于处理器间的线程漂移（threads drifting）导致性能统计波动



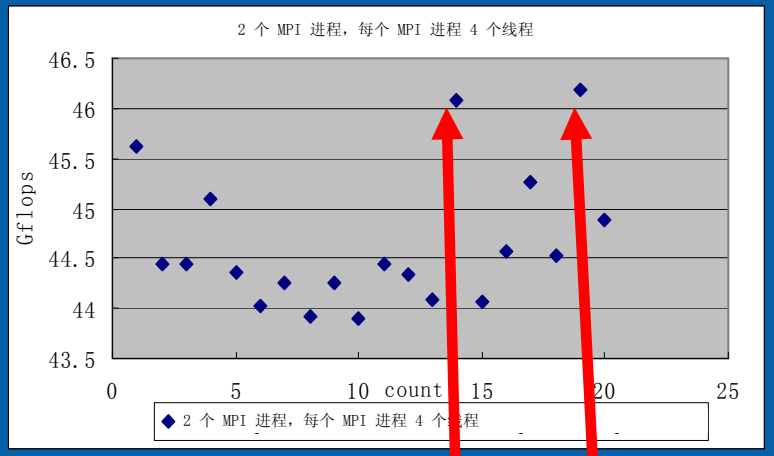


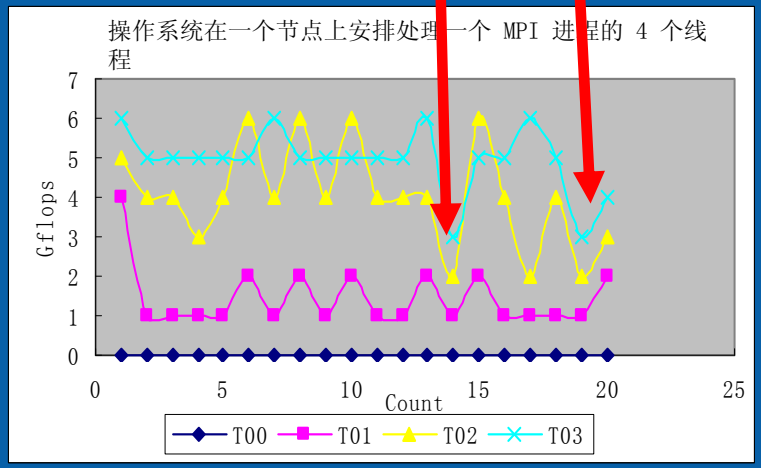
- 在每个线程处理应用程序中，性能均有统计波动。
- 除非将线程绑定到处理器内核，否则波动是不可避免的
- 运行时处理器间存在线程跳跃



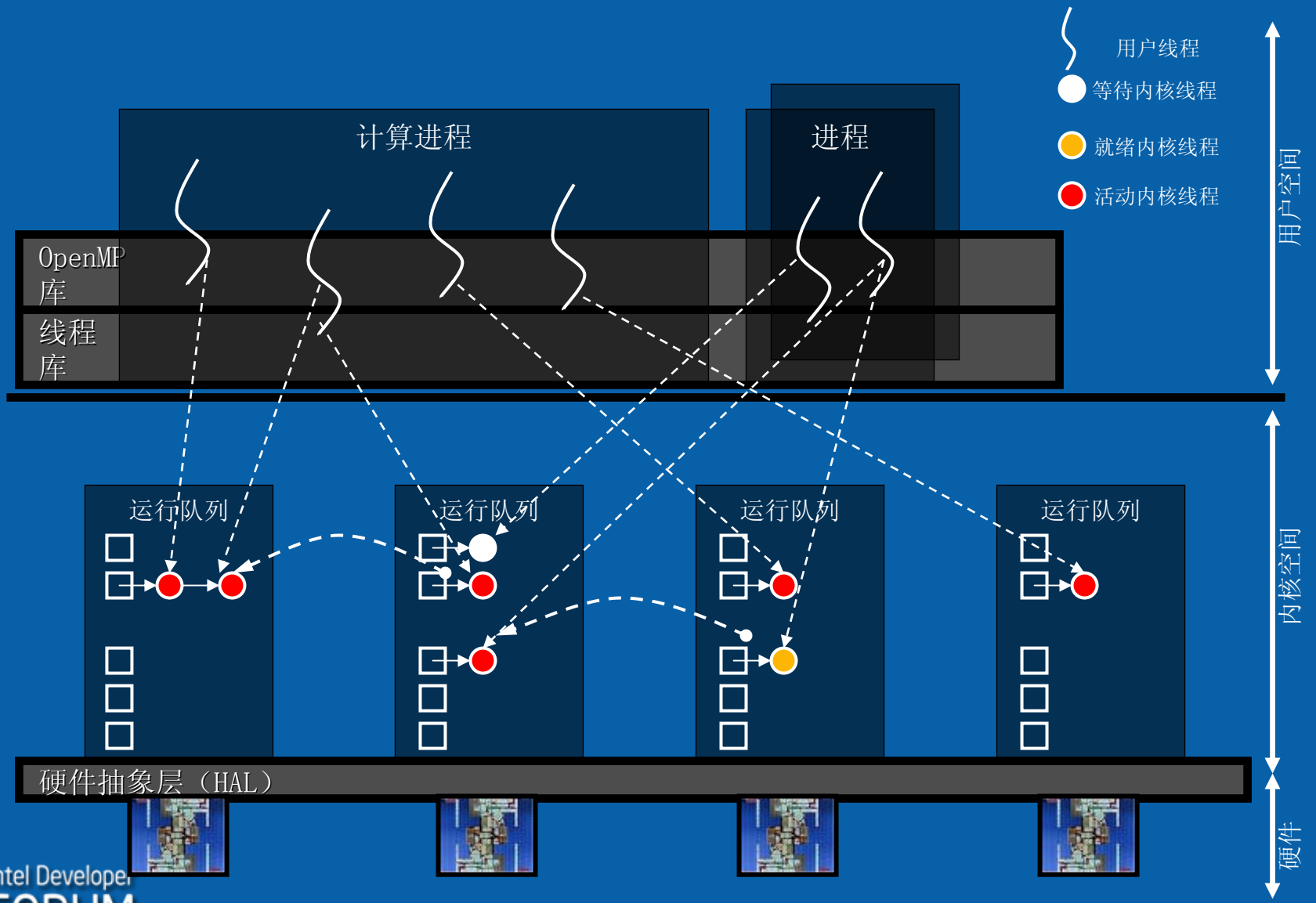
操作系统线程调度检测到的线程偏移

- 如果将 MPI 进程绑定到内核 0 和内核 7，则其它线程将由操作系统调度。
- T00 到 T03 由编号为 1 的 MPI 进程创建。T00 是主线程。
- 当MPI进程的线程创建在同一个插槽上时，则可在 Xeon 5300 平台上获得最佳性能





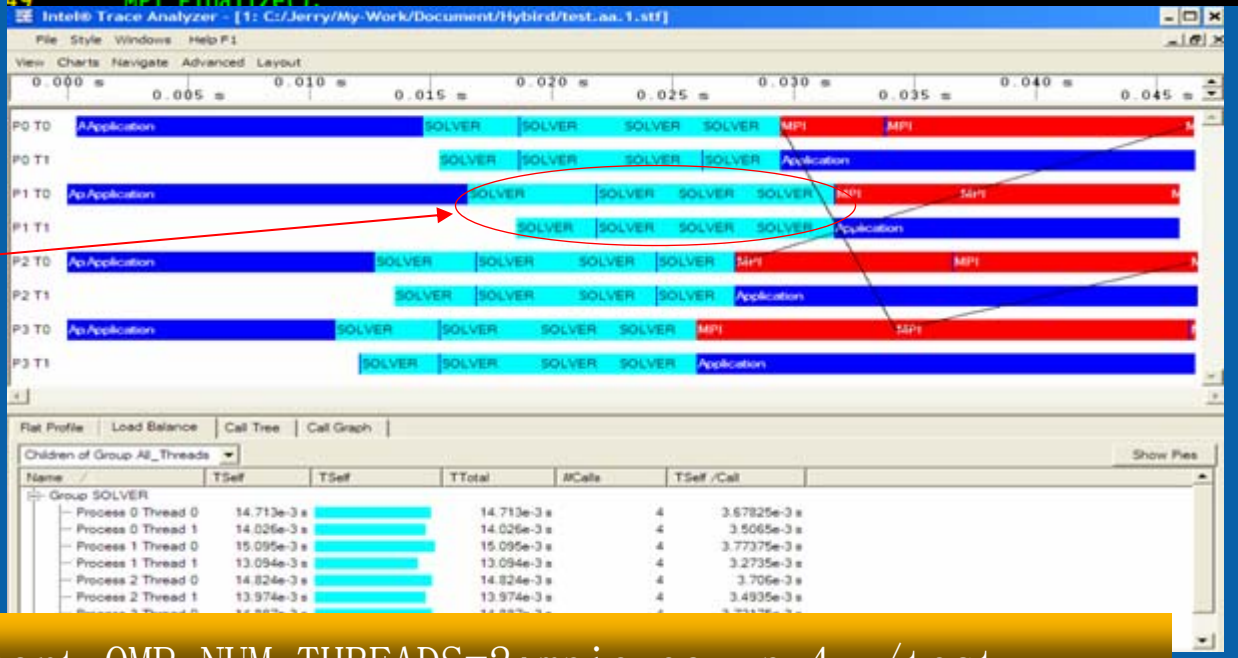
Linux 2.6 中的 OpenMP 线程



ITAC 支持通过手动添加探测代码跟踪 OpenMP 线程

- 第 1 步： 在函数的源代码中插入函数 VT_funcdef/VT_enter/VT_leave，然后在线程中执行这些函数
- 第 2 步： 编译源代码时链接到英特尔® 跟踪分析器工具库
- 第 3 步： 运行应用程序。
- 第 4 步： 应用程序运行完成后，通过英特尔® 跟踪分析器工具观察跟踪数据。
- 注意线程处理部分
- 注意插入的探测代码带来的开销

```
20 //        Set VT enter
21          VT_funcdef("SOLVER:execute",VT_NOCLASS,&statehandle);
22          VT_enter(statehandle,VT_NOSCL);
23 +-- 13 行: for (k=0;k<size;k++) {----------
36 }
37          if ( myid%2 == 0 ) {
38              MPI_Send(x,1000000,MPI_FLOAT,dest,100,MPI_COMM_WORLD);
39                      MPI_Recv(x,1000000,MPI_FLOAT,source,100,MPI_COMM_WORLD,&stat);
40          } else {
41                      MPI_Recv(x,1000000,MPI_FLOAT,source,100,MPI_COMM_WORLD,&stat);
42              MPI_Send(x,1000000,MPI_FLOAT,dest,100,MPI_COMM_WORLD);
43          }
44           }
45 //        clear VT enter
46          VT_leave(VT_NOSCL);
47 //
48
49      MPI_Finalize();
```

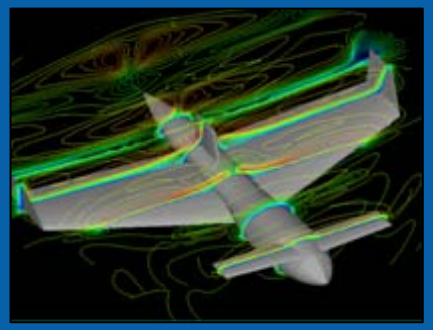


Export OMP_NUM_THREADS=2;mpiexec –n 4 ./test

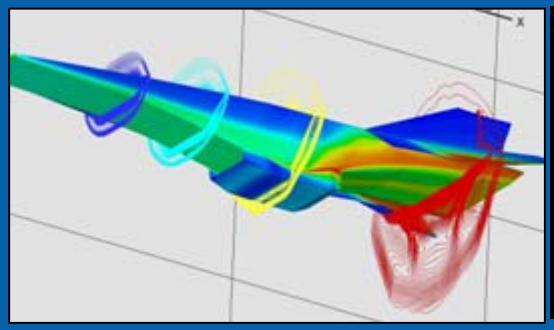
Intel Developer FORUM

NAPA 混合并行：调试 MPI 和 OpenMP

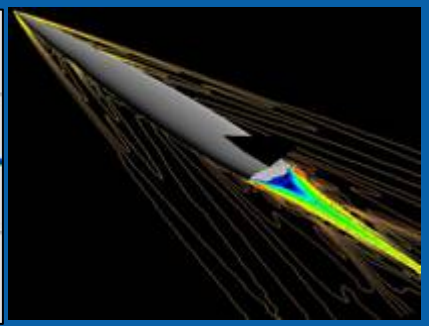
- Napa* 是 NUAA 自己开发的 CFD 应用程序，英特尔已对其代码进行了彻底的 MPI 优化
- MPI 版本有许多占对点通信，可用于在每次迭代时更新影子网格（ghost grid）的数据，并且集中通信量不多
- MPI 版本在以太网上的速度提升不到 60，可扩展能力较低。这是应用混合并行模型的一个很好的试验对象



MAV



X-43



NASA 升力式航天器

* NAPA - NuAa Program for Aerodynamics（NuAa 空气动力学程序），由 NUAA 开发，并经过英特尔/NUAA 的优化

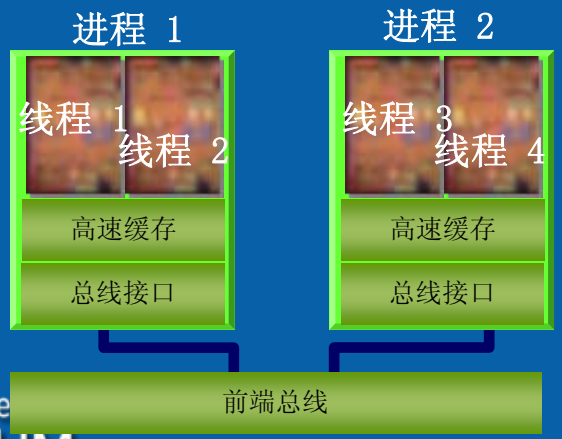
MPI 调试：同时考虑计算和通信

- 计算与通信重叠？
  - 对于共享存储、以太网、InfiniBand 和 Numa Link 中的一般 MPI 实现（Mpich1.2、英特尔 mpi 1.0），答案是否定的。
  - 对于某些特定的消息传递结构，答案是肯定的。
    - 例如： QCDOC 超级计算机及其 LQCD 消息传递 API（QMP）
  - 针对集群的 NAPA 并行处理已经完成，并通过性能指标评测结果进行了验证。

OpenMP 与 MPI 简单比较

- 在采用至强 5100 的 1 个节点（含 4 个内核）之上进行的性能指标评测结果
  - 通过英特尔编译器编译的 mpich 1.2.7p1
  - 英特尔 Fortran 编译器 9.1.033

加速 \ 内核数量	1 个内核	2 个内核	4 个内核
串行	1	-	-
OpenMP	-	1.77	2.8
MPI	-	1.76	3.14（4 个 MPI 进程）
混合	-	-	3.10（2 个线程/进程）



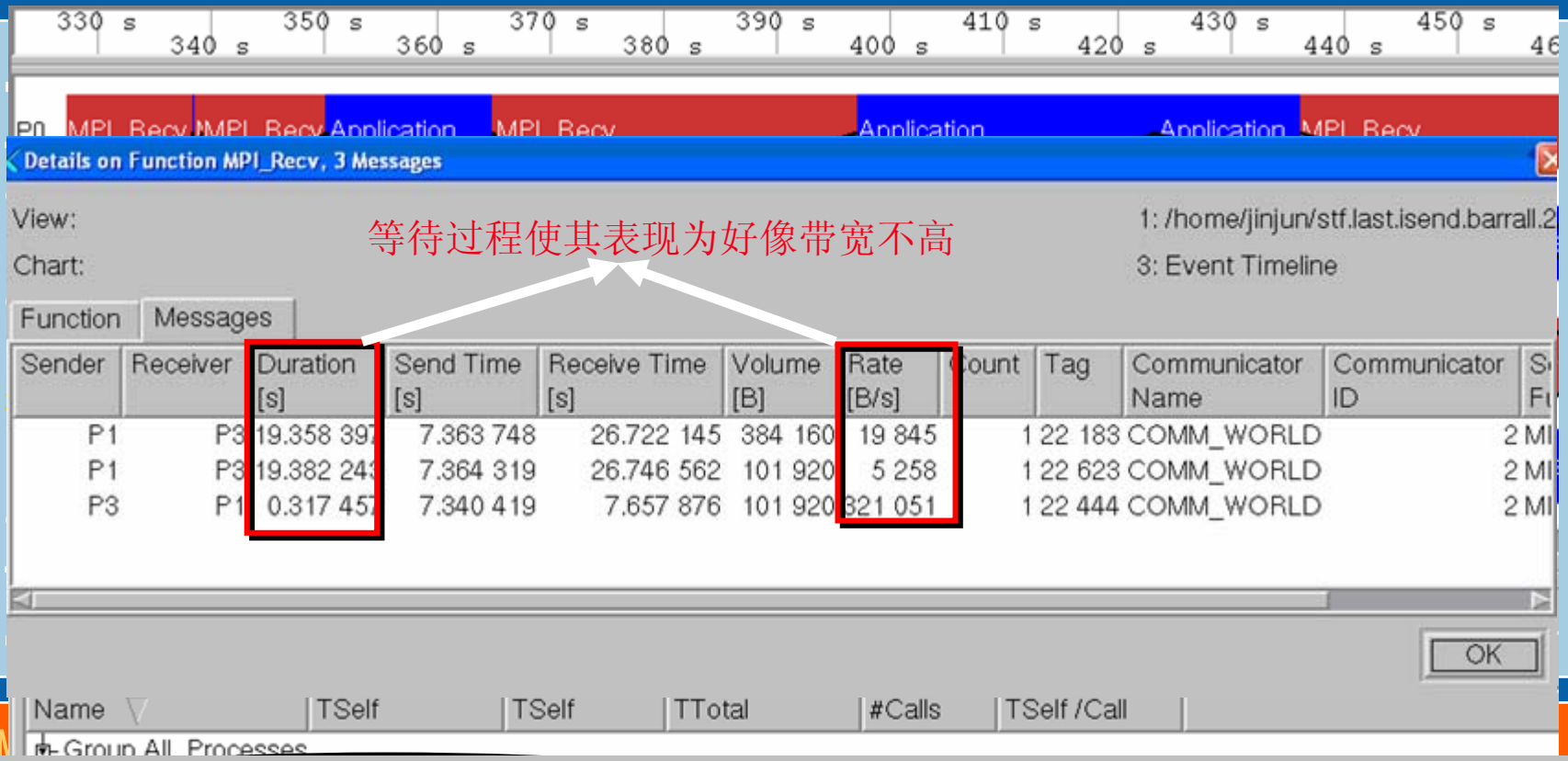
混合模型

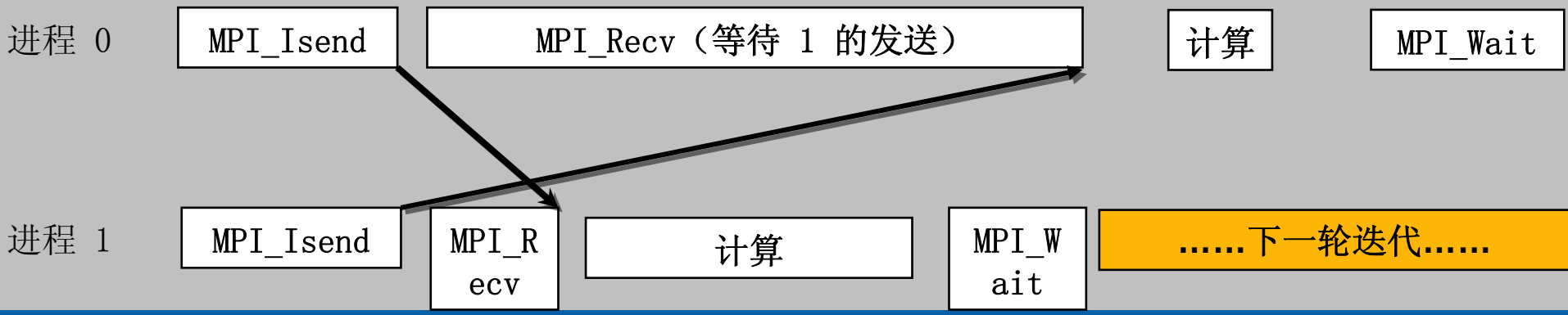
OpenMP 不能够很好地支持从 2 个内核到 4 个内核的扩展，其开销 $T_{th}>T_{mpi}$

MPI 在 4 个内核时性能更高

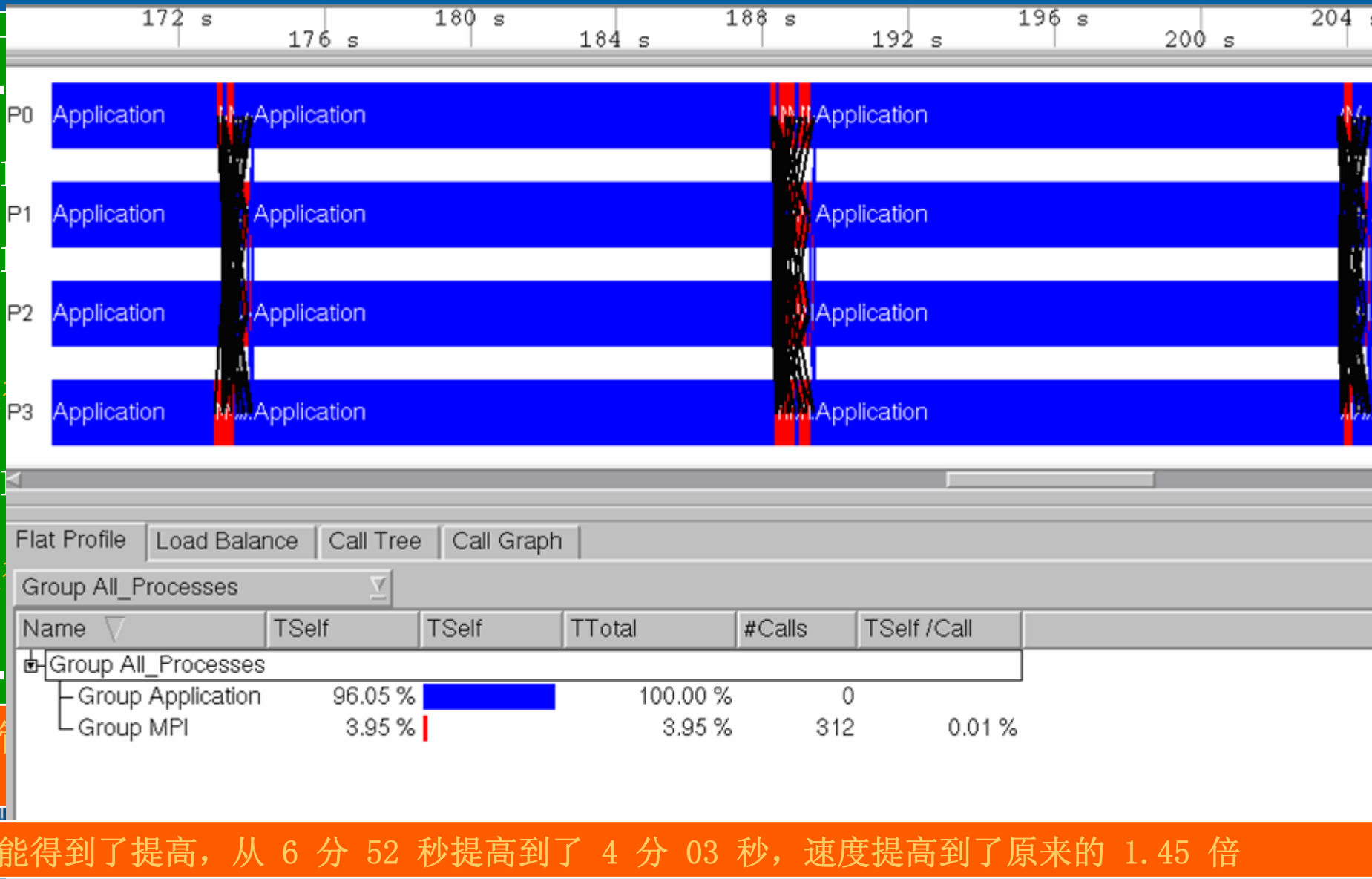
NAPA： MPI 通信和计算

计算网格：97*97*97*24



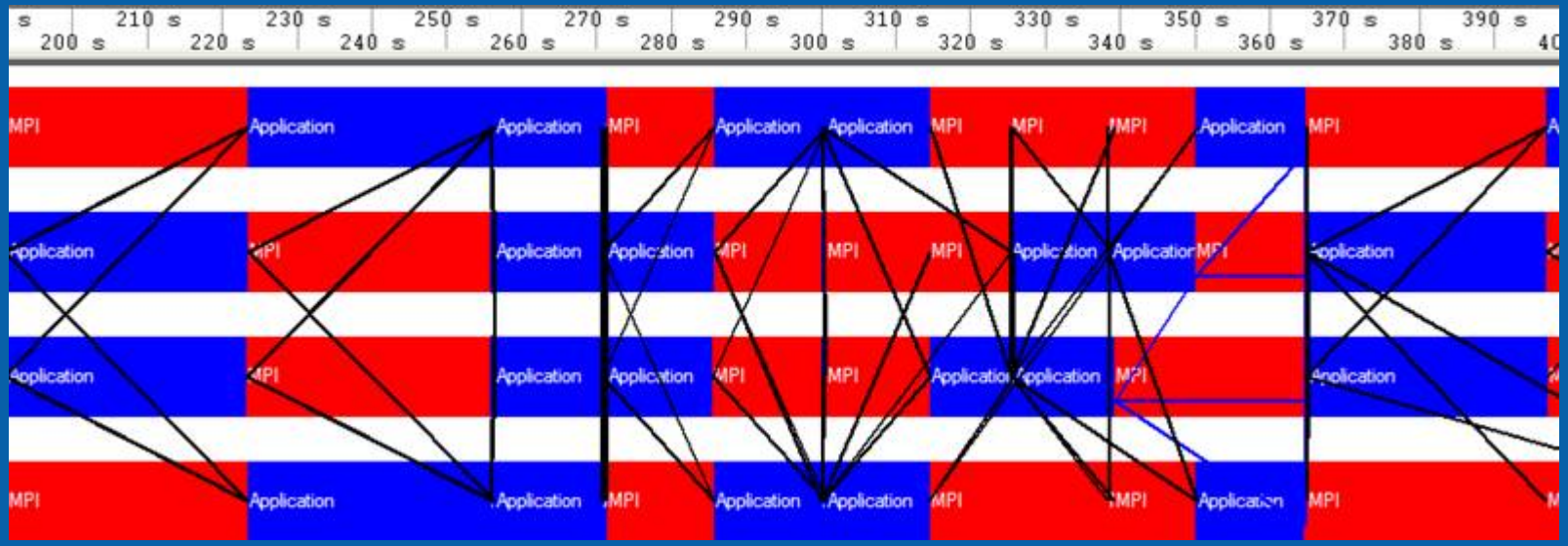


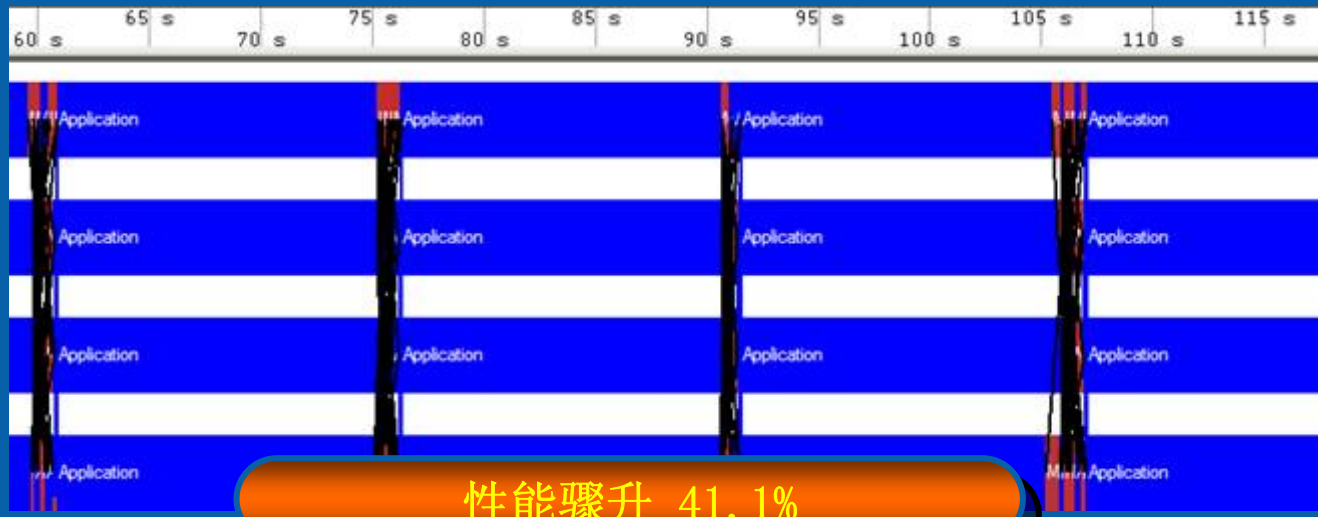
MPI 通信和计算（续）



性能得到了提高，从 6 分 52 秒提高到了 4 分 03 秒，速度提高到了原来的 1.45 倍

NAPA： MPI 并行优化应用于实际的工作负载



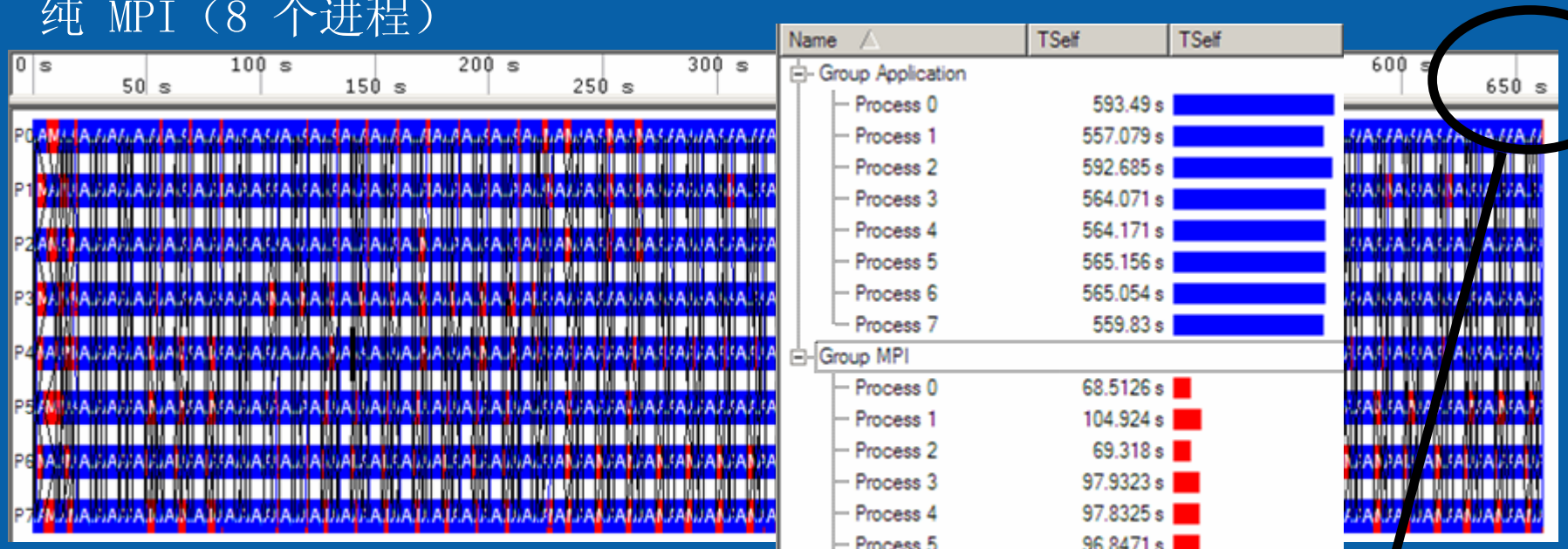


工作负载：超音速进气道4进程模拟

性能骤升 41.1%

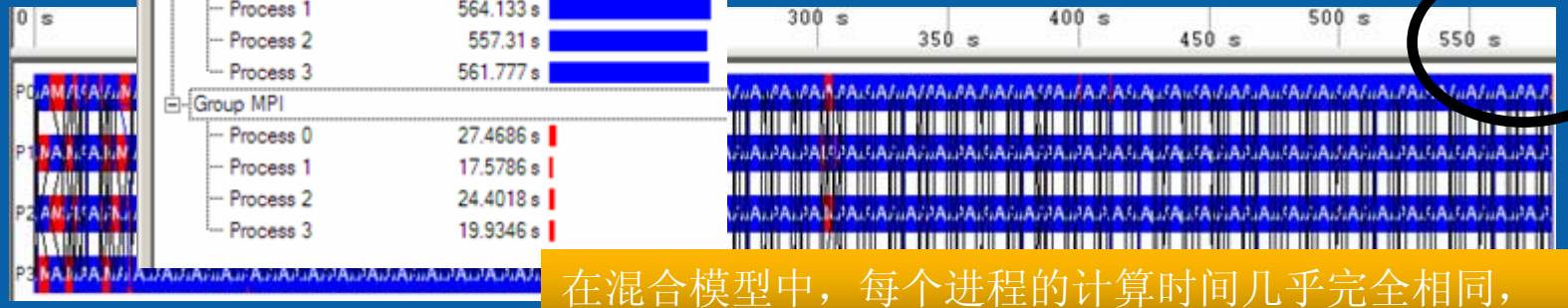
线程减少了计算量

纯 MPI（8 个进程）



Name	TSelf	TSelf
Group Application		
Process 0	593.49 s	
Process 1	557.079 s	
Process 2	592.685 s	
Process 3	564.071 s	
Process 4	564.171 s	
Process 5	565.156 s	
Process 6	565.054 s	
Process 7	559.83 s	
Group MPI		
Process 0	68.5126 s	
Process 1	104.924 s	
Process 2	69.318 s	
Process 3	97.9323 s	
Process 4	97.8325 s	
Process 5	96.8471 s	
Process 6	96.9494 s	
Process 7	102.173 s	

混合（[illegible]进程）

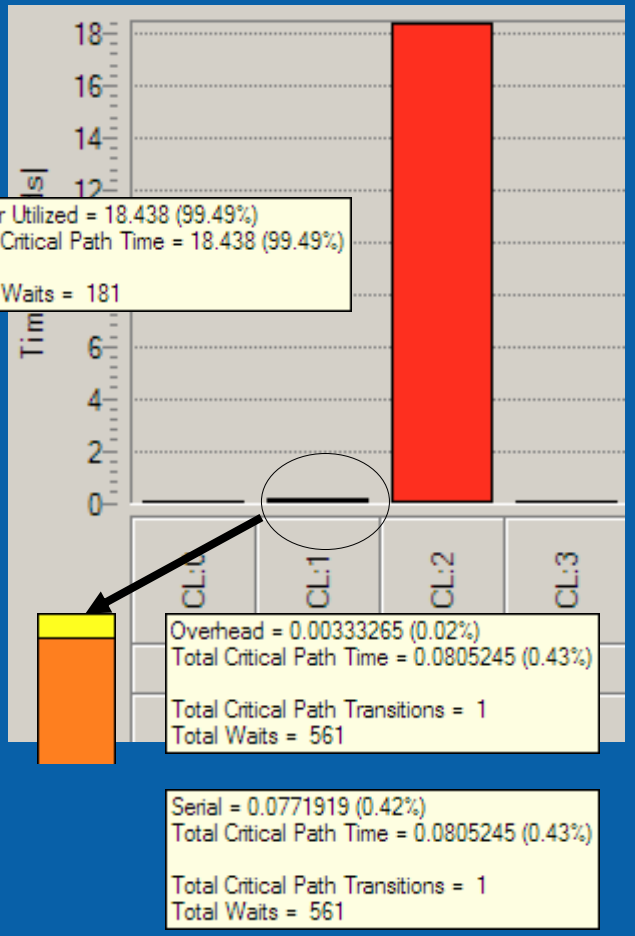


Name	TSelf	TSelf
Group Application		
Process 0	554.243 s	
Process 1	564.133 s	
Process 2	557.31 s	
Process 3	561.777 s	
Group MPI		
Process 0	27.4686 s	
Process 1	17.5786 s	
Process 2	24.4018 s	
Process 3	19.9346 s	

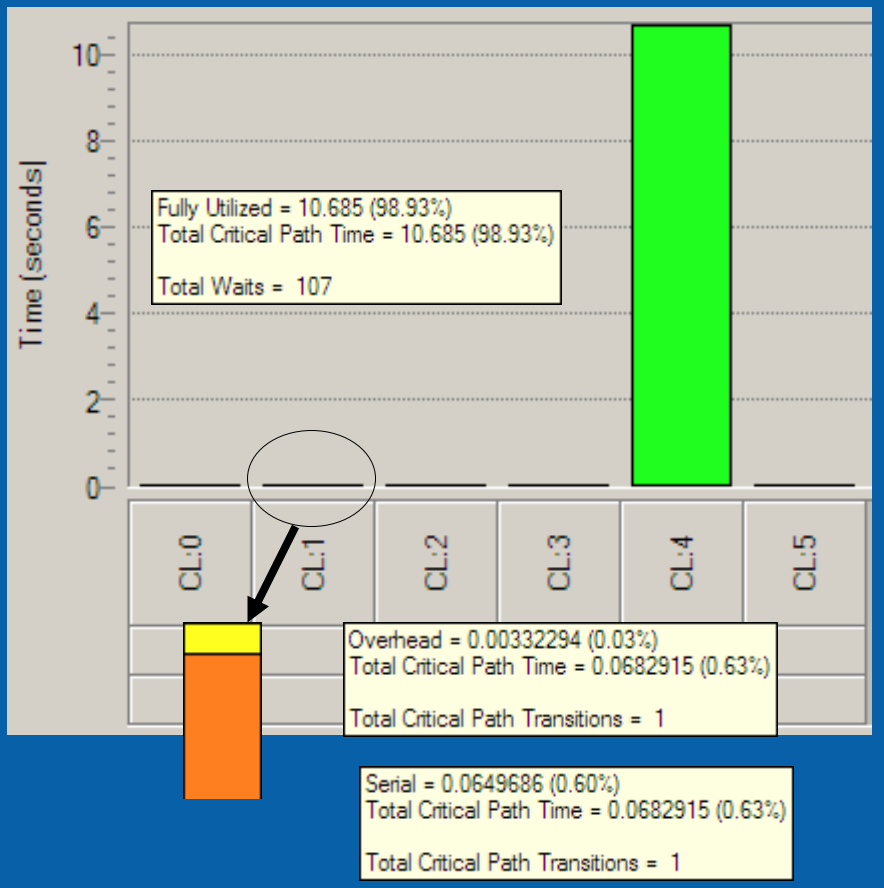
在混合模型中，每个进程的计算时间几乎完全相同，
通信量大幅减少，从 13.9%（纯 MPI）减少到 3.84%（混合）

Intel Developer FORUM

通过英特尔线程调节器调节 OpenMP 线程



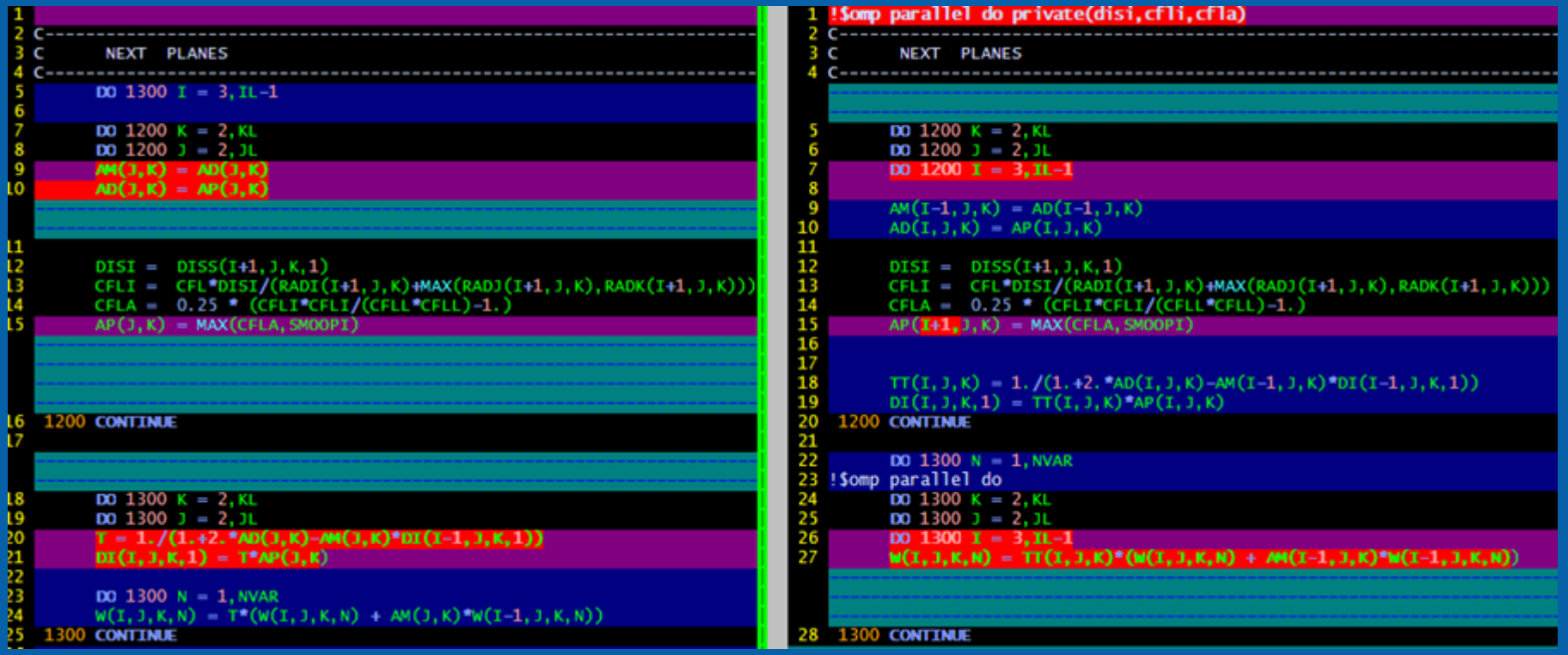
2 个线程



4 个线程

Intel Developer FORUM

进行 MPI 调试后，避免简单化OpenMP – 对其进行优化



```
C-----------------------------------------------------------------------
C       NEXT  PLANES
C-----------------------------------------------------------------------
      DO 1300 I = 3,IL-1

      DO 1200 K = 2,KL
      DO 1200 J = 2,JL
      AM(J,K) = AD(J,K)
      AD(J,K) = AP(J,K)

      DISI =  DISS(I+1,J,K,1)
      CFLI =  CFL*DISI/(RADI(I+1,J,K)+MAX(RADJ(I+1,J,K),RADK(I+1,J,K)))
      CFLA =  0.25 * (CFLI*CFLI/(CFLL*CFLL)-1.)
      AP(J,K) = MAX(CFLA,SMOOPI)

 1200 CONTINUE

      DO 1300 K = 2,KL
      DO 1300 J = 2,JL
      T = 1./(1.+2.*AD(J,K)-AM(J,K)*DI(I-1,J,K,1))
      DI(I,J,K,1) = T*AP(J,K)

      DO 1300 N = 1,NVAR
      W(I,J,K,N) = T*(W(I,J,K,N) + AM(J,K)*W(I-1,J,K,N))
 1300 CONTINUE
```

```
!$omp parallel do private(disi,cfli,cfla)
C-----------------------------------------------------------------------
C       NEXT  PLANES
C-----------------------------------------------------------------------

      DO 1200 K = 2,KL
      DO 1200 J = 2,JL
      DO 1200 I = 3,IL-1

      AM(I-1,J,K) = AD(I-1,J,K)
      AD(I,J,K) = AP(I,J,K)

      DISI =  DISS(I+1,J,K,1)
      CFLI =  CFL*DISI/(RADI(I+1,J,K)+MAX(RADJ(I+1,J,K),RADK(I+1,J,K)))
      CFLA =  0.25 * (CFLI*CFLI/(CFLL*CFLL)-1.)
      AP(I+1,J,K) = MAX(CFLA,SMOOPI)


      TT(I,J,K) = 1./(1.+2.*AD(I,J,K)-AM(I-1,J,K)*DI(I-1,J,K,1))
      DI(I,J,K,1) = TT(I,J,K)*AP(I,J,K)
 1200 CONTINUE

      DO 1300 N = 1,NVAR
!$omp parallel do
      DO 1300 K = 2,KL
      DO 1300 J = 2,JL
      DO 1300 I = 3,IL-1
      W(I,J,K,N) = TT(I,J,K)*(W(I,J,K,N) + AM(I-1,J,K)*W(I-1,J,K,N))


 1300 CONTINUE
```

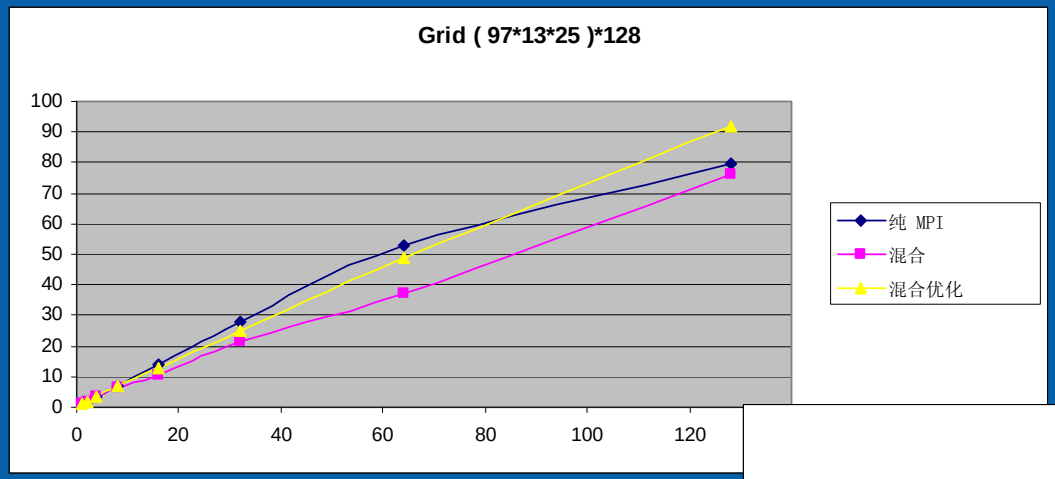
调试 OpenMP 部分，提高其计算并行粒度，降低循环中的数据相关性

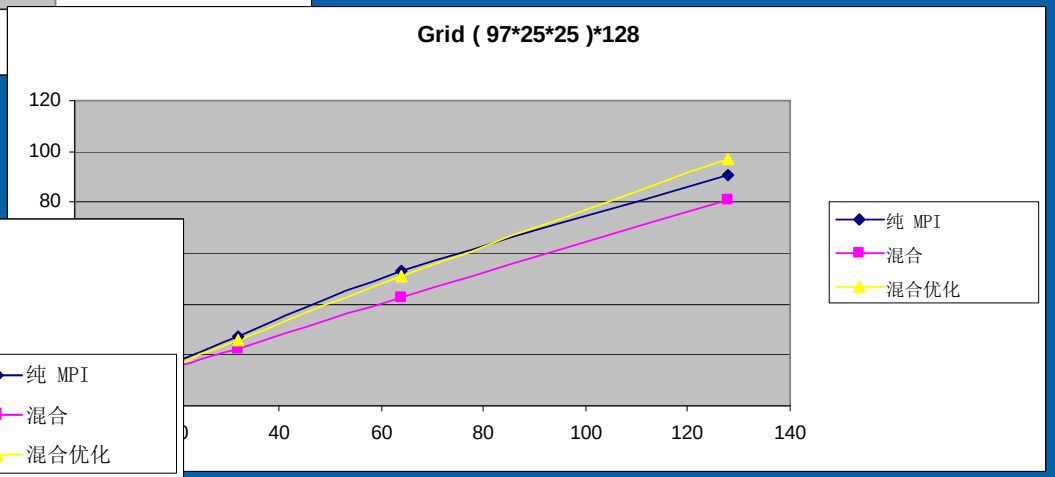
Intel Developer FORUM

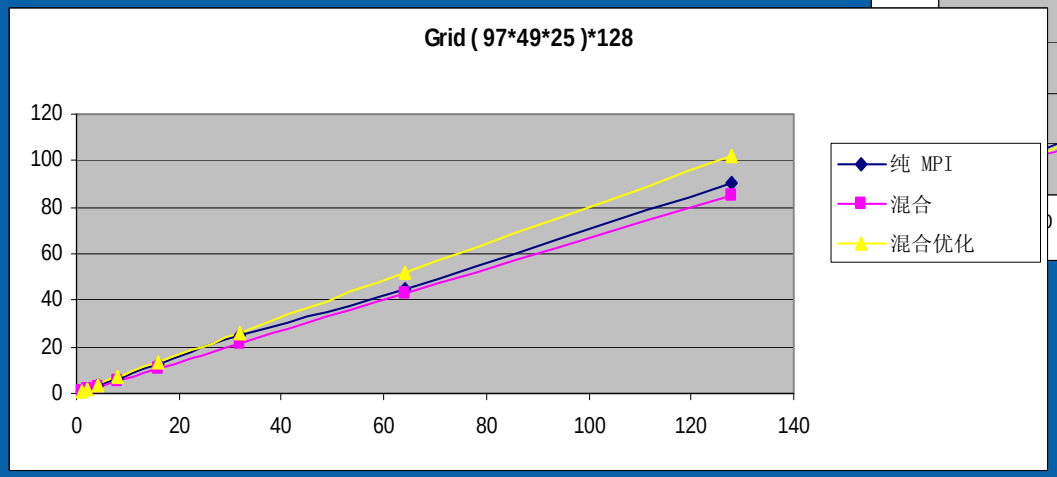
性能指标评测结果： 混合模型在MPI基础上进一步提升性能

混合： 2 个线程获得约 1.6 倍的加速比

混合优化： 2 个线程获得约 1.77 倍的加速比



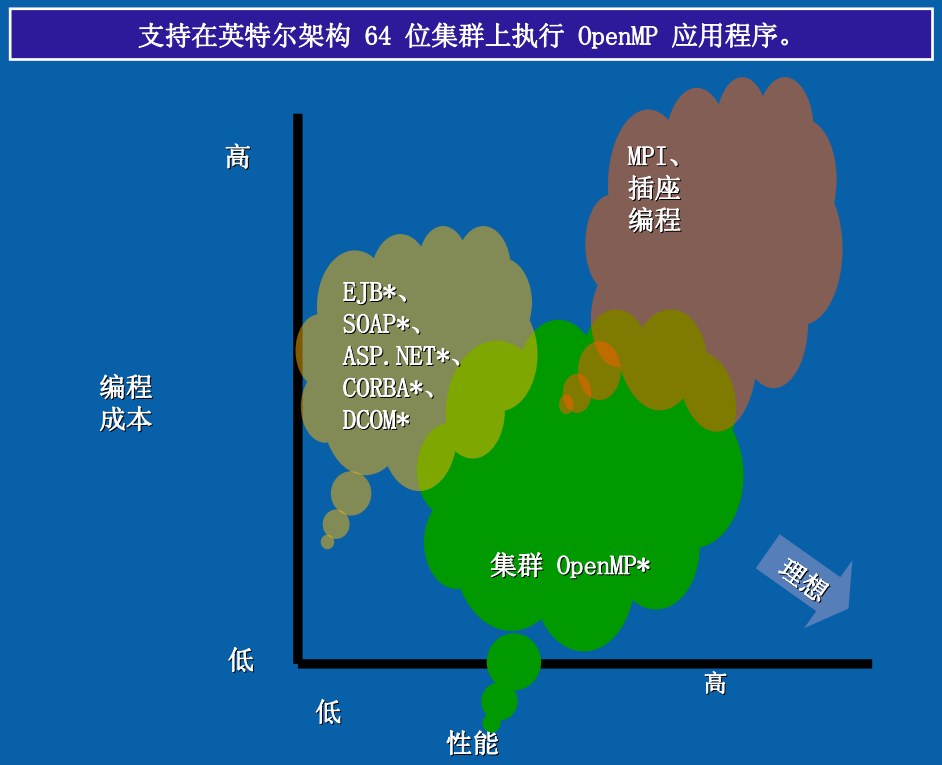




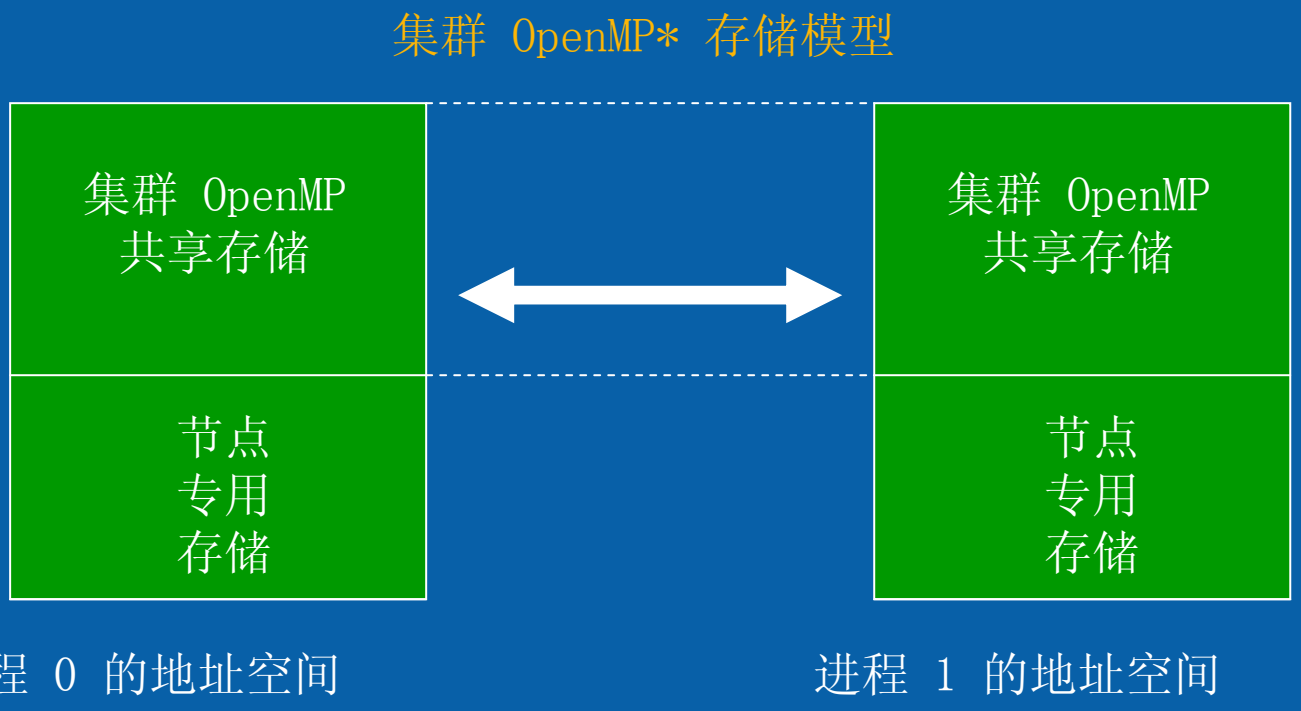
- 对于混合模型，每个进程包含两个线程
- MPI 性能指标评测在相同的节点时实现了共享存储

以“混合”模型形式出现的英特尔Cluster OpenMP：一种规避MPI实现复杂性的工具

- 英特尔Cluster OpenMP 是一种 OpenMP 扩展，可跨集群节点部署OpenMP 线程。
- Cluster OpenMP* 是一种较易实施的集群编程方式。有些代码可通过集群 OpenMP 获得良好的性能和扩展。将代码移植用 Cluster OpenMP 是一种系统化过程，比将进程移植到消息传递的工作量要少得多。可以使用性能工具帮助优化程序，在集群中获得最佳性能。

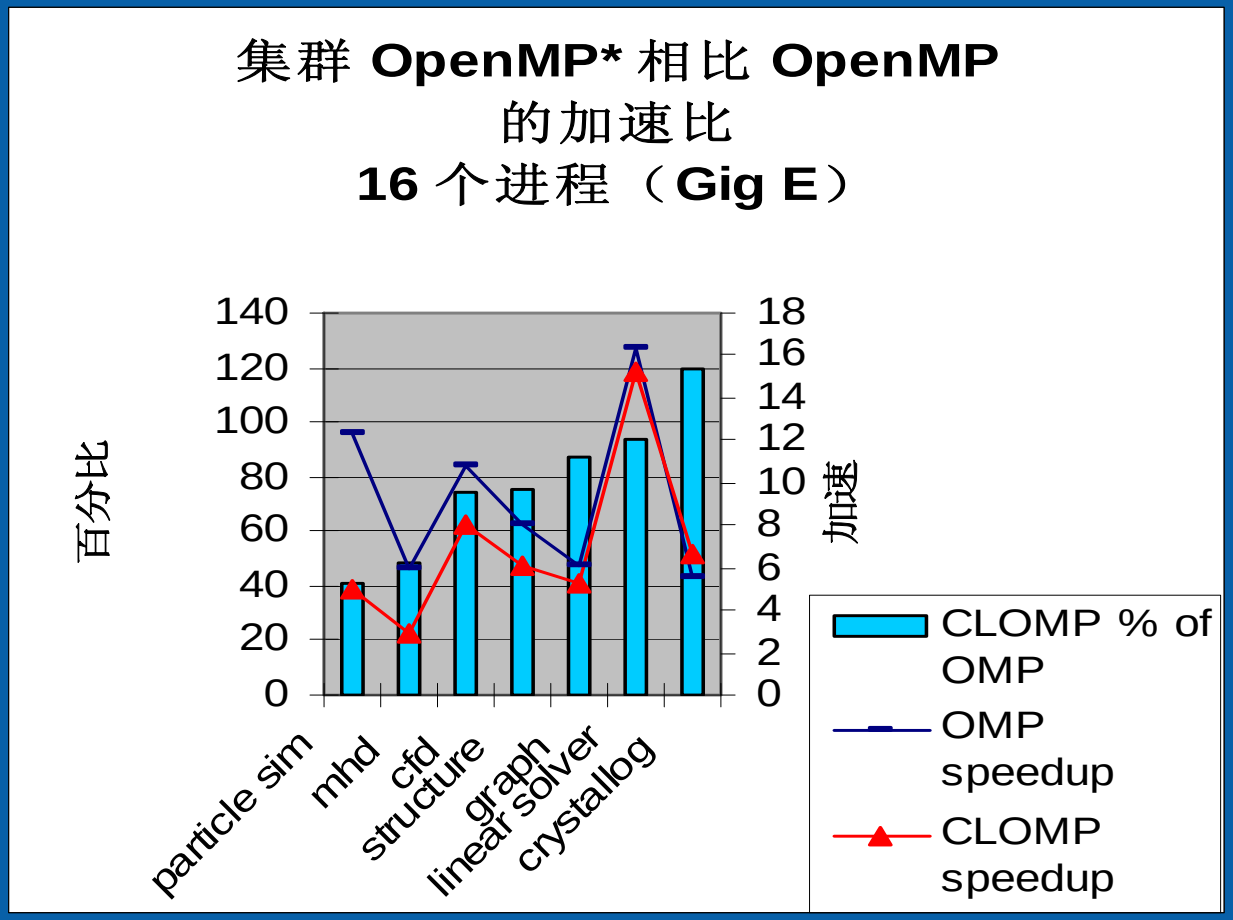


集群 OpenMP 工作原理



集群 OpenMP 是通过编译器扩展和一个运行时库实现的，该运行时库支持在集群上运行 OpenMP 程序。集群 OpenMP 存储模型可以在分布式节点间创建一个虚拟共享存储区域，该区域可方便用户避免 MPI 编程，同时充分利用分布式存储。

集群 OpenMP 在混合架构中的可扩展性

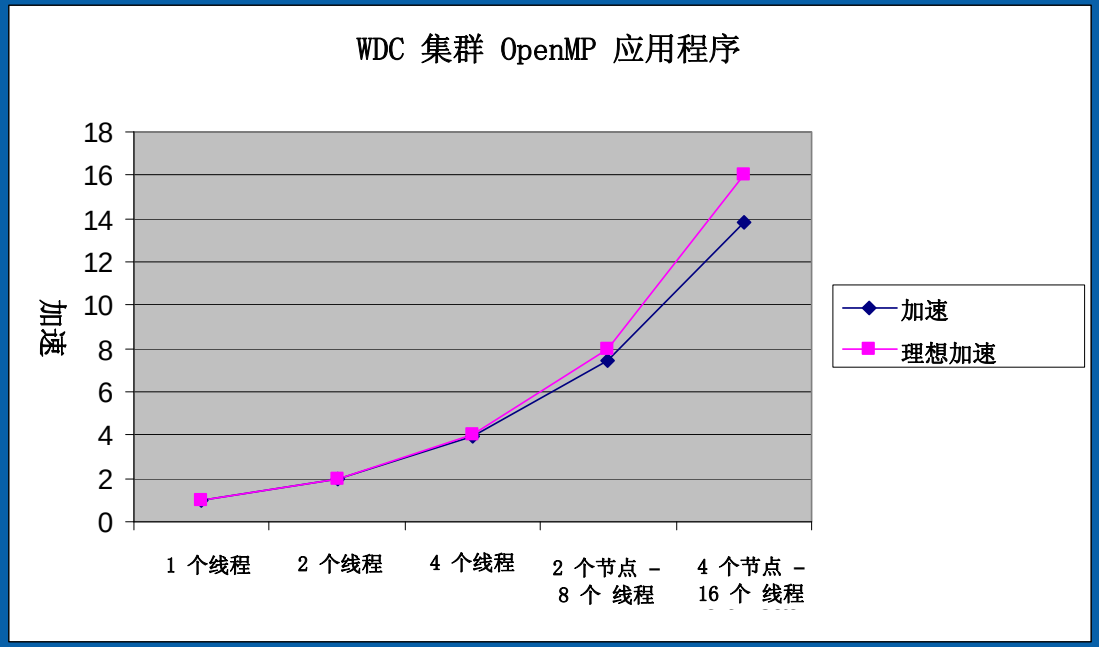


*文中涉及的其它名称及商标属于各自所有者资产。

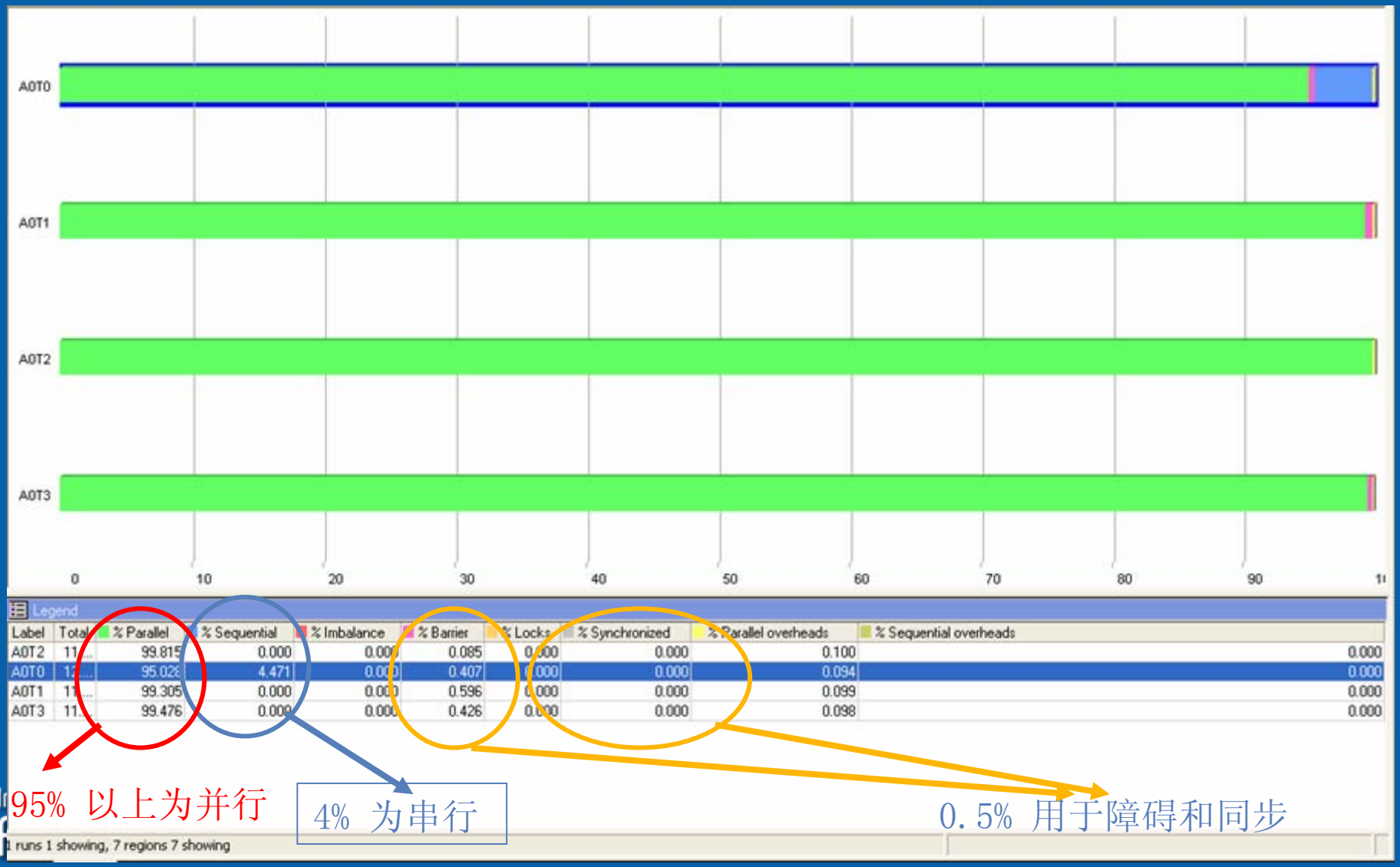
案例研究： 双路至强 5100 混合集群中的 GalRen

- 主函数： 渲染一段空间飞行的影片。
- 该函数使用了一个星体数据库（包含星体的亮度、颜色、位置等数据），并根据这些数据进行光线跟踪。
- 每个线程处理帧缓冲的一组扫描线，并计算像素。

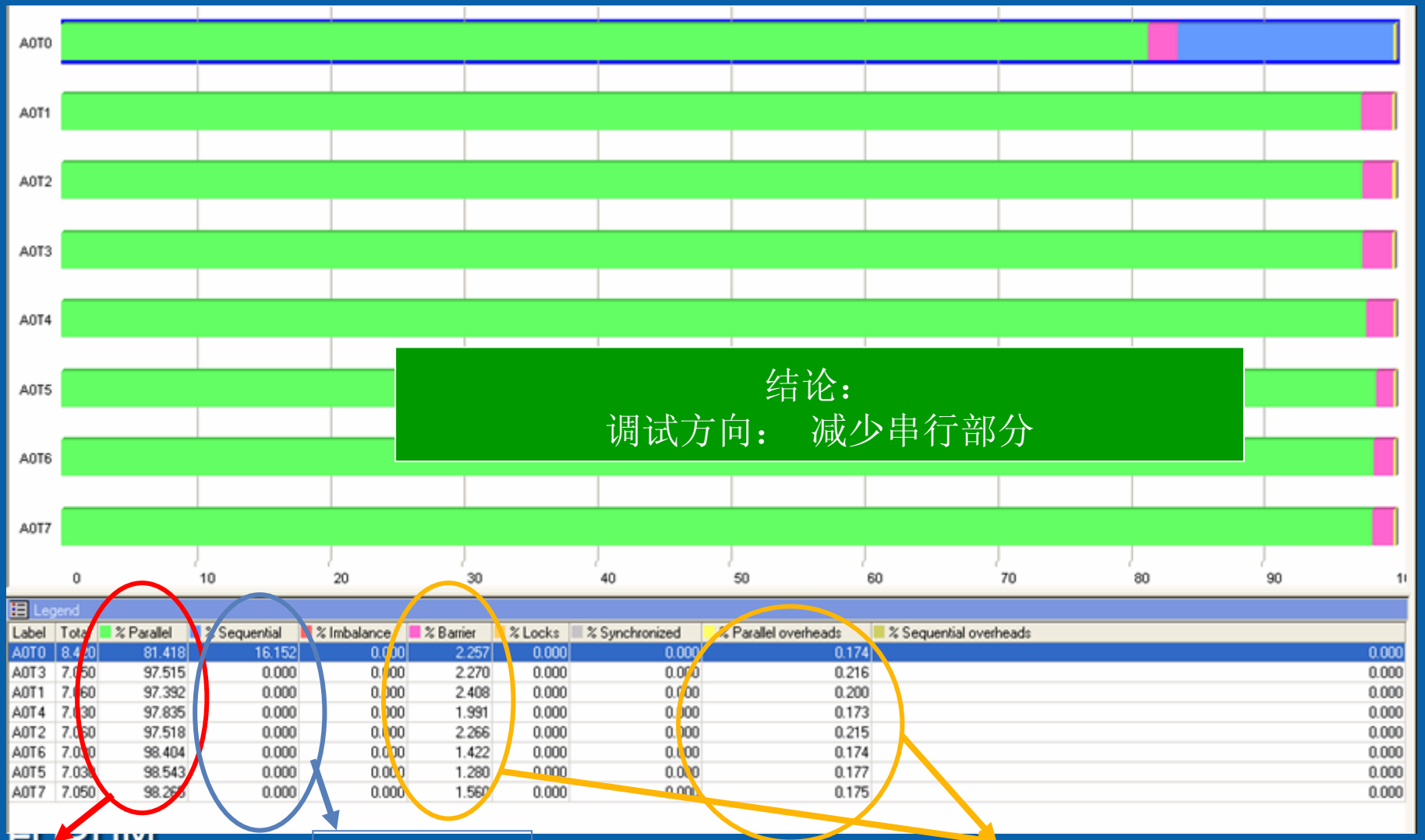
16 个线程的速度可提高 14 倍



案例研究： GalRen：一个节点中有 4 个线程



2 个节点中有 8 个线程



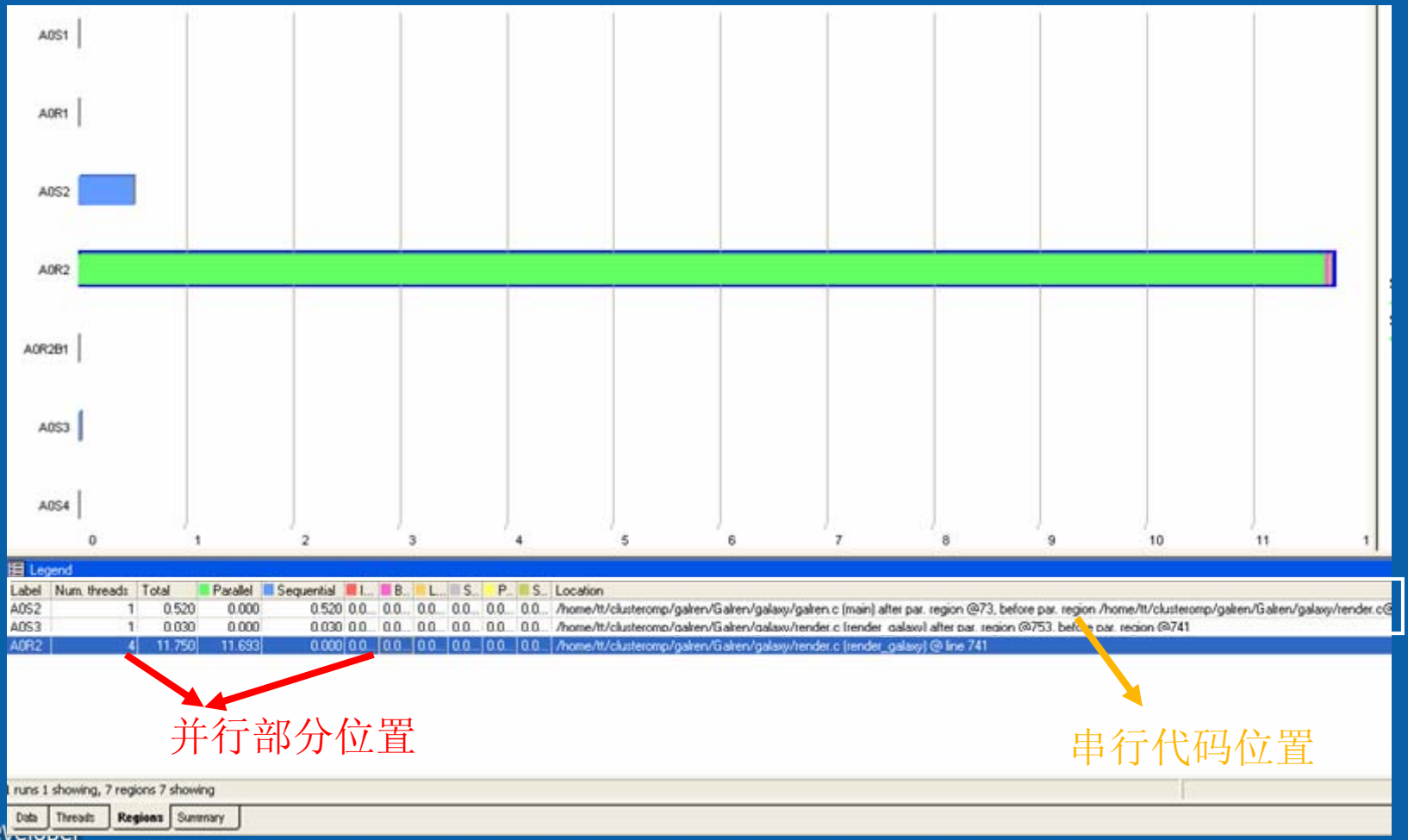
81% 以上为并行

16% 为串行

2.5% 用于障碍和同步

如何找出源代码：位置与线程

一个节点中有 4 个线程



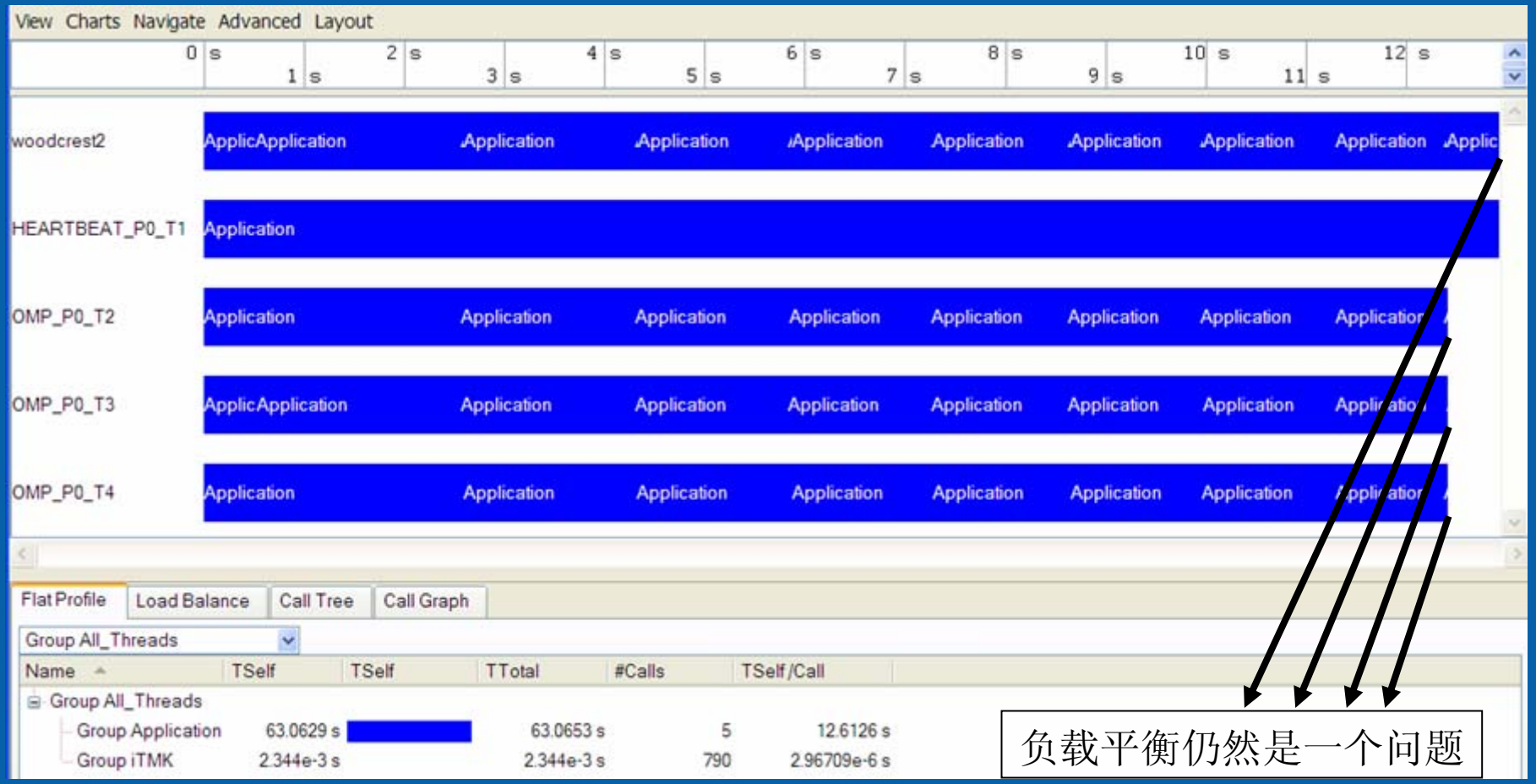
```
#pragma omp parallel private(i, j, R, G, B, A)
    {
#if defined(OMP)
      int thread_num = omp_get_thread_num();
      int my_slice = xsize / omp_get_num_threads();
      int my_offset = (thread_num * my_slice);
      int my_bytes = ( sizeof( uByte8 ) * my_slice );
      uByte8 *my_pixels;

      my_pixels = malloc( my_bytes );
      memset( my_pixels, 0x55, my_bytes );
#endif
#pragma omp for schedule(runtime)
      for (j=0; j < ysize; j++) {
           GraphicsDrawRow( my_offset, ysize - j - 1, my_slice, my_pixels );
     for (i=0; i < xsize; i++) {
           ComputeRay( g, s, image, i, j);
        } /* end for index */
    GraphicsDrawRow( 0, ysize - j - 1, xsize, &image[xsize*j]);
    }
#if defined(OMP)
      free( my_pixels );
#endif
}
```

一部分集群 OpenMP 伪码

需要极少量的更改

ITC/ITA（4 个线程*1 个节点）

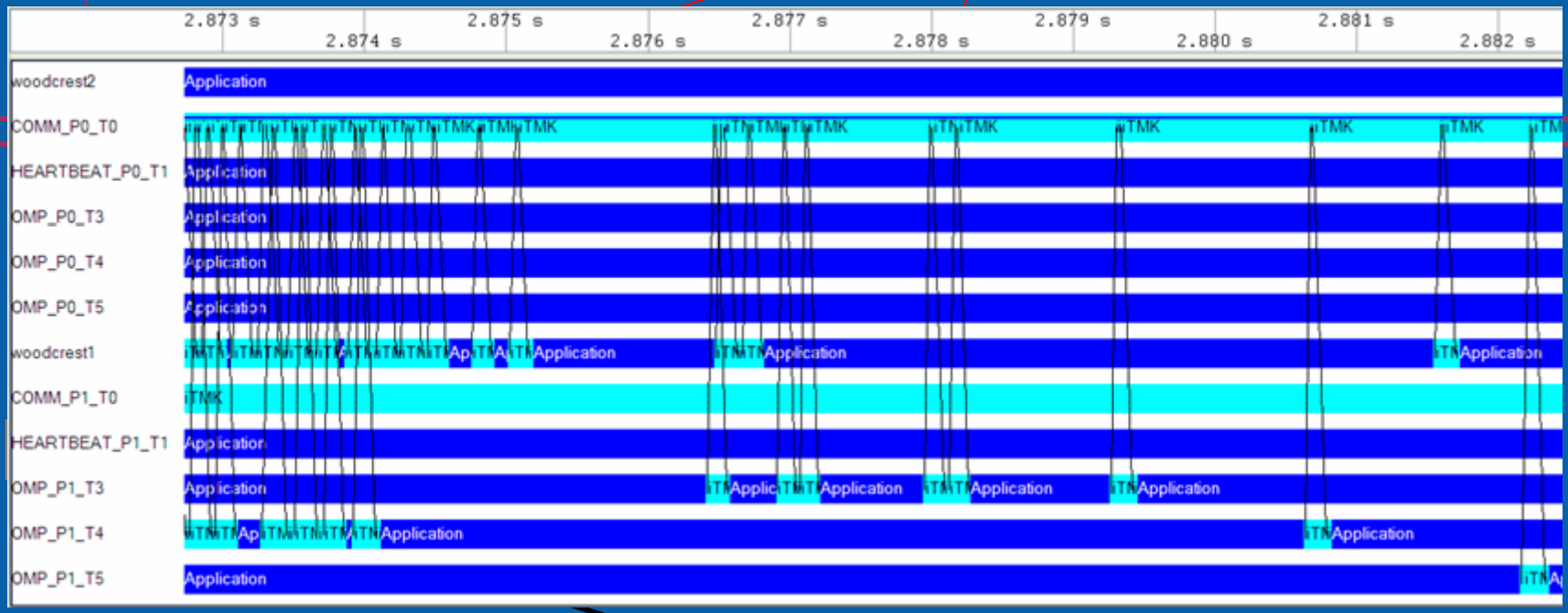


单个节点中几乎无通信部分

ITC/ITA（4 个线程*2 个节点）

*COMM_P*_T0 线程* 处理
与其它节点中的线程的通信

仅在节点间传递的消息

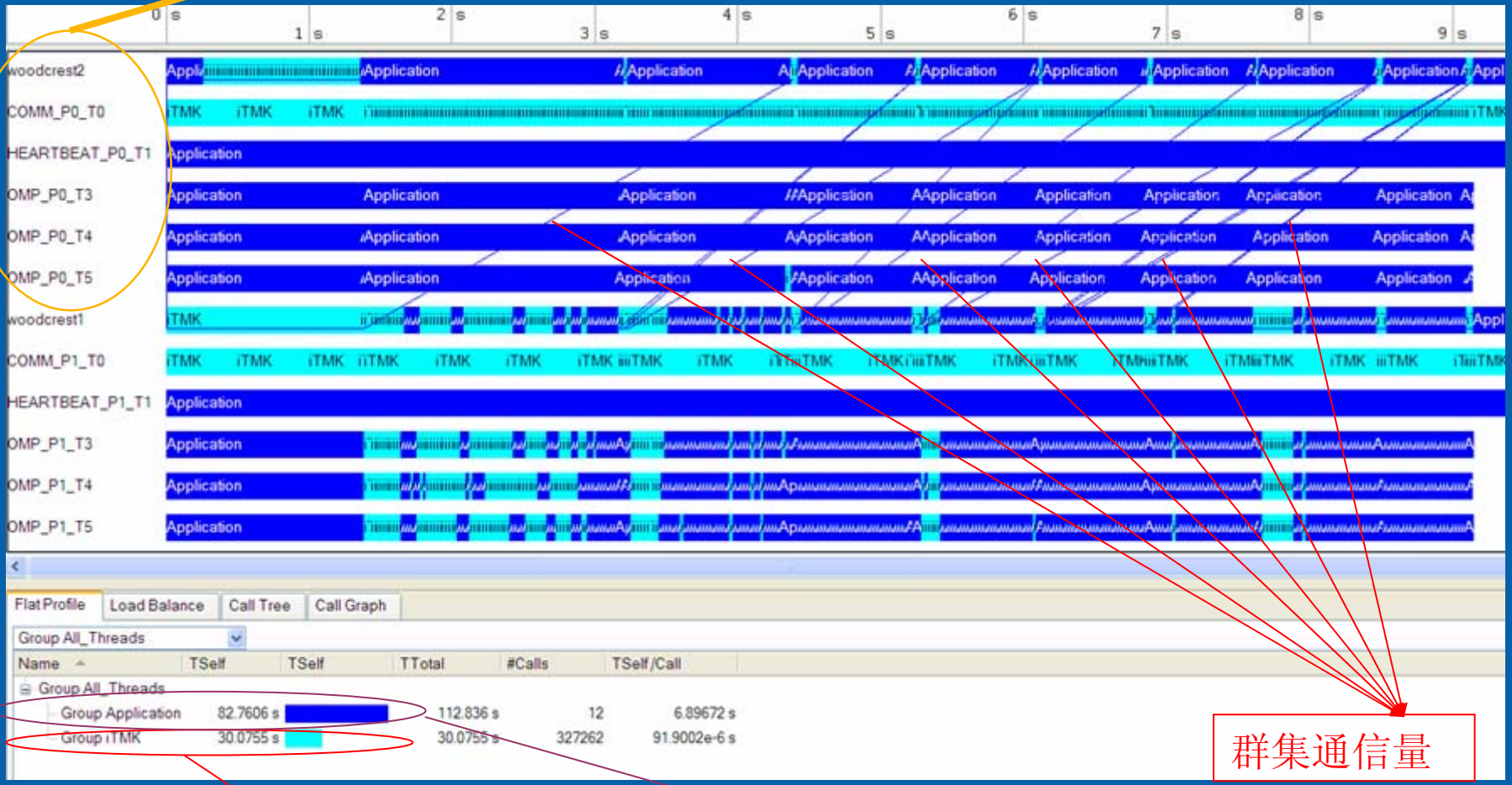


主线程 和
一个节点中用于计算的其它 *3 个次级线程*

每个节点的 *节拍线程*

ITC/ITA（4 个线程*2 个节点）

增加线程可以更好地处理通信和计算的重叠



总结/行动号召

- 同时考虑线程处理和MPI进程并行处理，当前的多核 SMP 集群中实现最大的可扩展性。
- 权衡使用MPI和OpenMP，利用英特尔 ITAC 和线程处理工具分析线程的行为和开销。
- 考虑并行粒度，在引入优化的 OpenMP 线程处理之前充分调试 MPI（大粒度）。
- 使用Cluster OpenMP 可以快速部署/调试 SMP 集群中的多线程处理，无需 MPI/PVM 编程即可使某些类型的应用程序获得相当大的速度提升。

请填写
课程评估表

非常感谢您的反馈，我们将据此改进未来的英特尔信息技术峰会。

欢迎参加秋季 IDF：
2007 年 9 月 18 -20 日 美国旧金山
2007 年 10 月 15 -16 日 中国台北

Intel Developer FORUM

- 感谢 ICSC CRT/HPC 支持团队，特别感谢 Wang Zhe、Qiao Nan、Jin Jun 和 Xu Jin。
- 还要感谢 Hoeflinger、Jay P、Meadows、Lawrence F 和 Ohlay、Patrick，感谢他们在集群 OpenMP 和 ITAC 线程处理跟踪问题上提供的帮助

Backup

集群 OpenMP 的优势

- 对于用户
  - 扩大问题规模；
  - 降低计算成本；
  - 消除能力瓶颈。
- 对于开发人员
  - 减少开发时间；
  - 提高可维护性；
  - 可以利用现有的 OpenMP 代码；
  - 易于实现，无需掌握 MPI 和 PVM 的知识；
  - 代码可读性好，与串行代码的一致性比 MPI/PVM 代码高；
  - 调试简单，采用小步进方法。

解决方案：

方法 1：

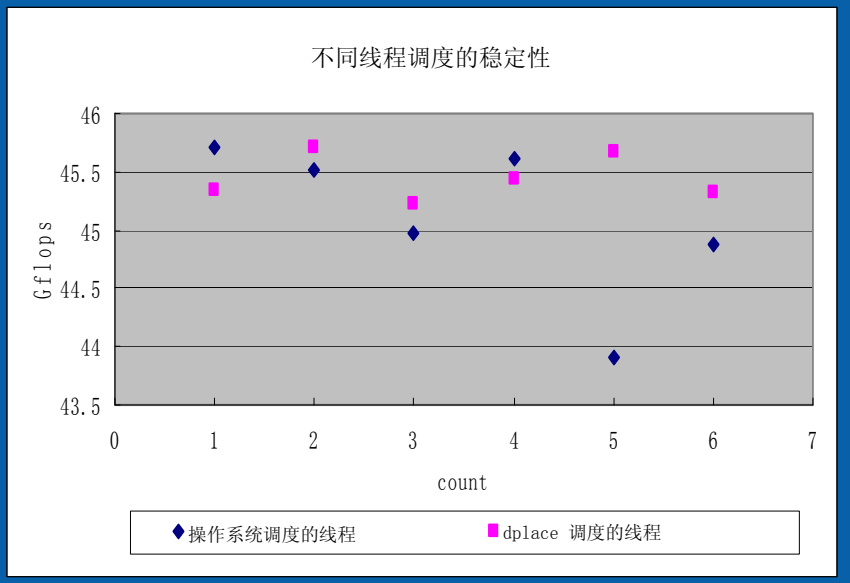
有些操作系统可以提供一些工具，允许在不同的内核|CPU上调度线程|进程， 例如 SGI 或 BULL 的 NUMA 工具。

方法 2：

在 OpenMP 区域中插入 OpenMP 函数 sched_setffinity， 例如：

```
#pragma omp parallel
{
     unsigned long mask = (1 << omp_get_thread_num()) *2 );

sched_setaffinity(0,sizeof(mask), &mask);
      source code
}
```



对 SGI Altix 进行性能指标评测。使用 dplace 调度线程