



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

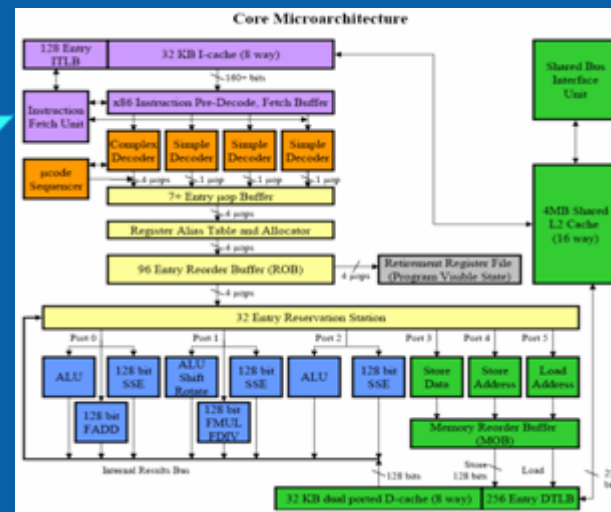
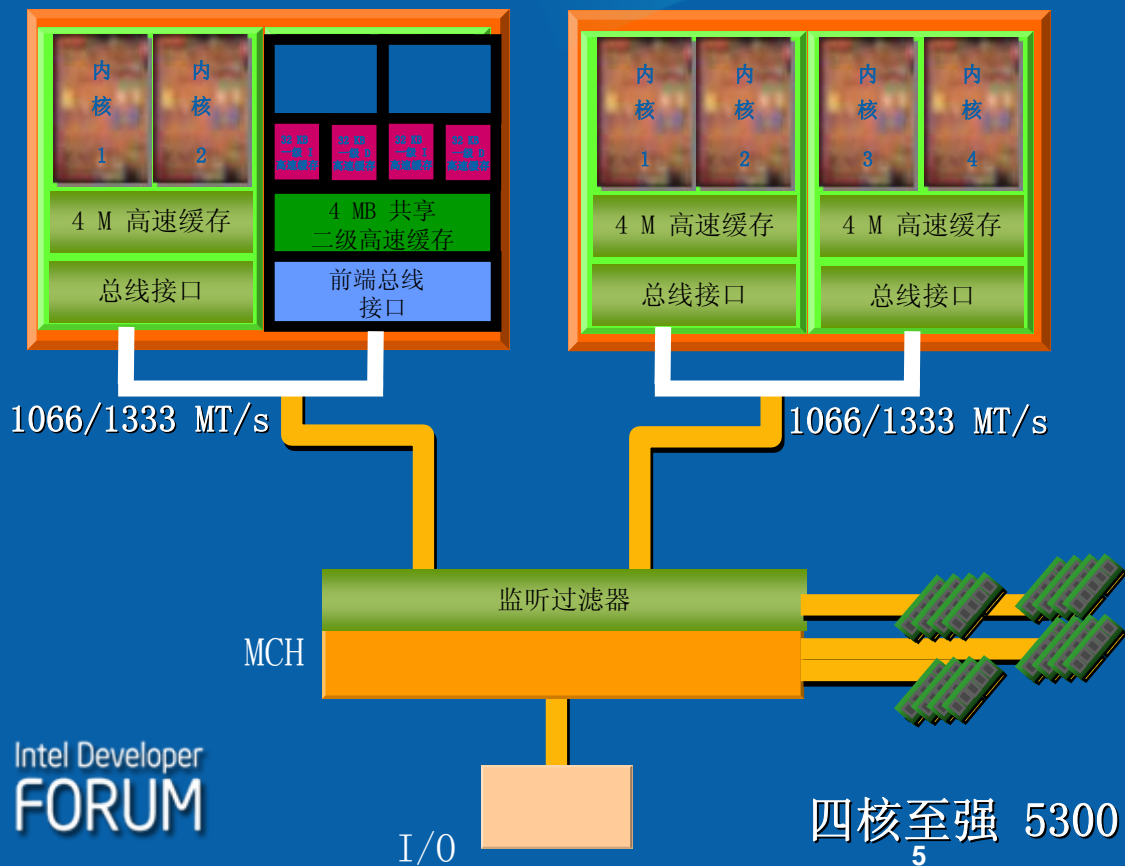
免责声明

- 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。
- 版权所有 © 2007 英特尔公司。所有权利受到保护。

议程

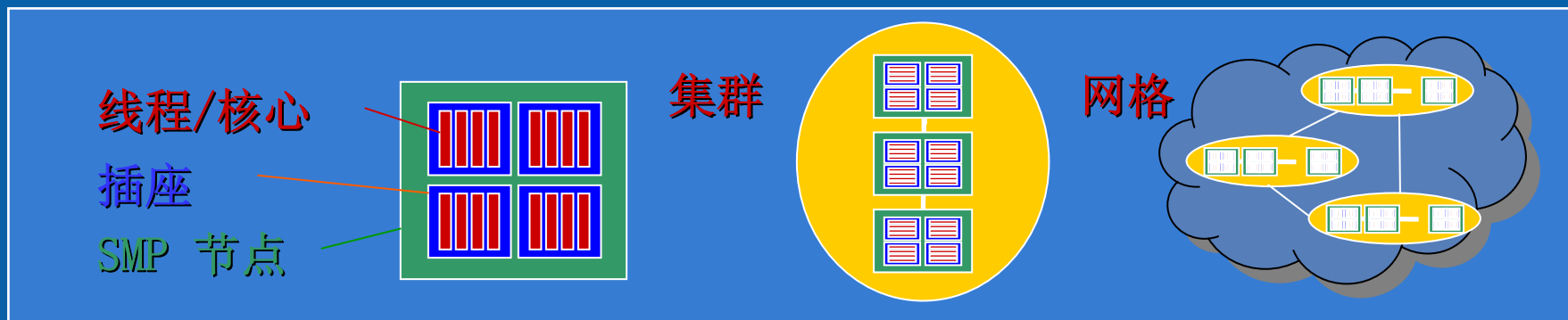
- 定义混合并行性能调试模型
- 对 HPL 进行性能调试：通过 ITAC 追踪线程抖动
- MPI 和 OpenMP 调试：对真实CFD应用的混合并行
- 尝试使用 Cluster-OpenMP 快速部署集群应用
- 总结/行动号召

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。

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	计算网络 /
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* 网络编程环境 (GPE)

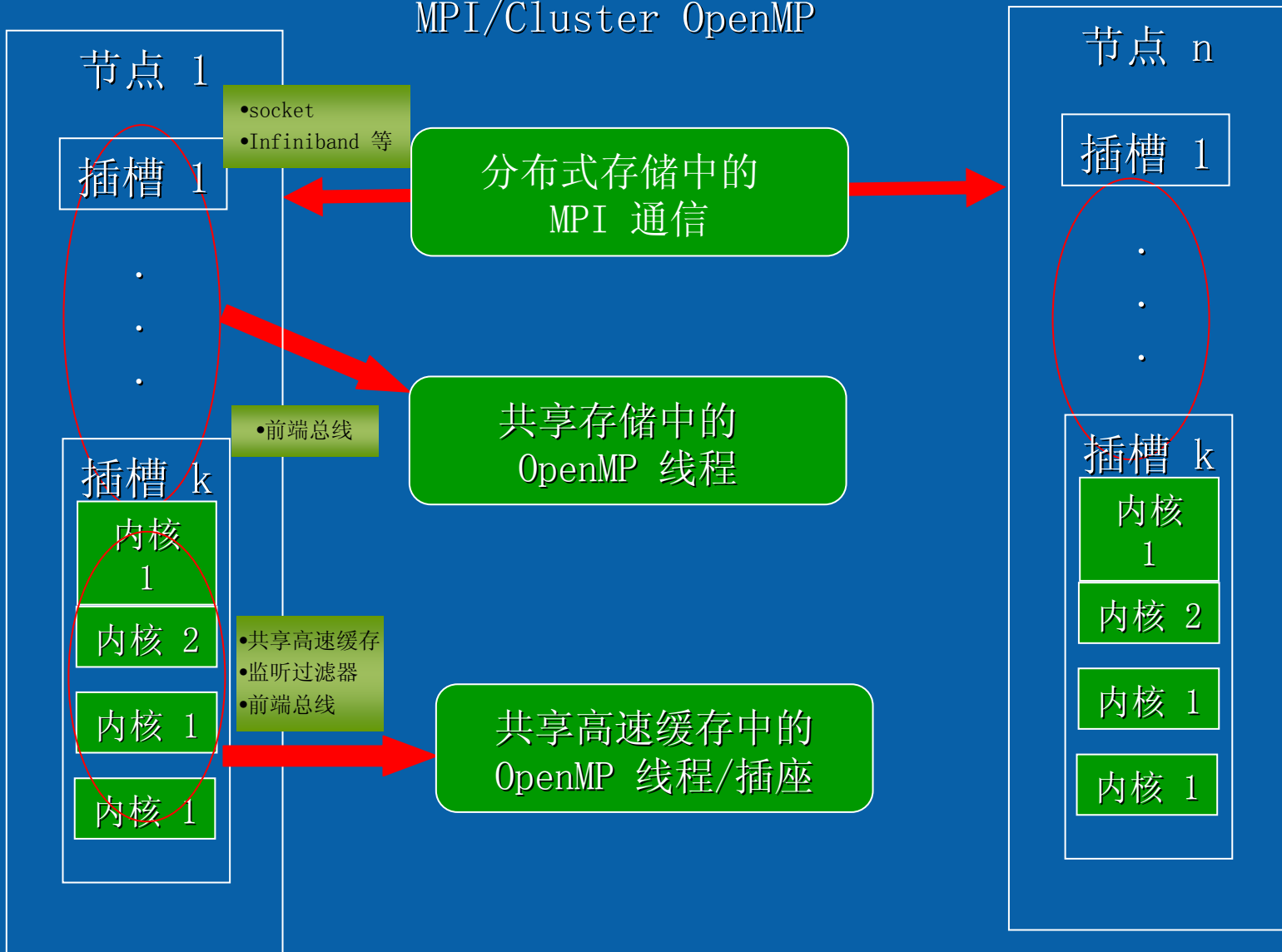
多核 SMP 集群模型

MPI/Cluster OpenMP

一级

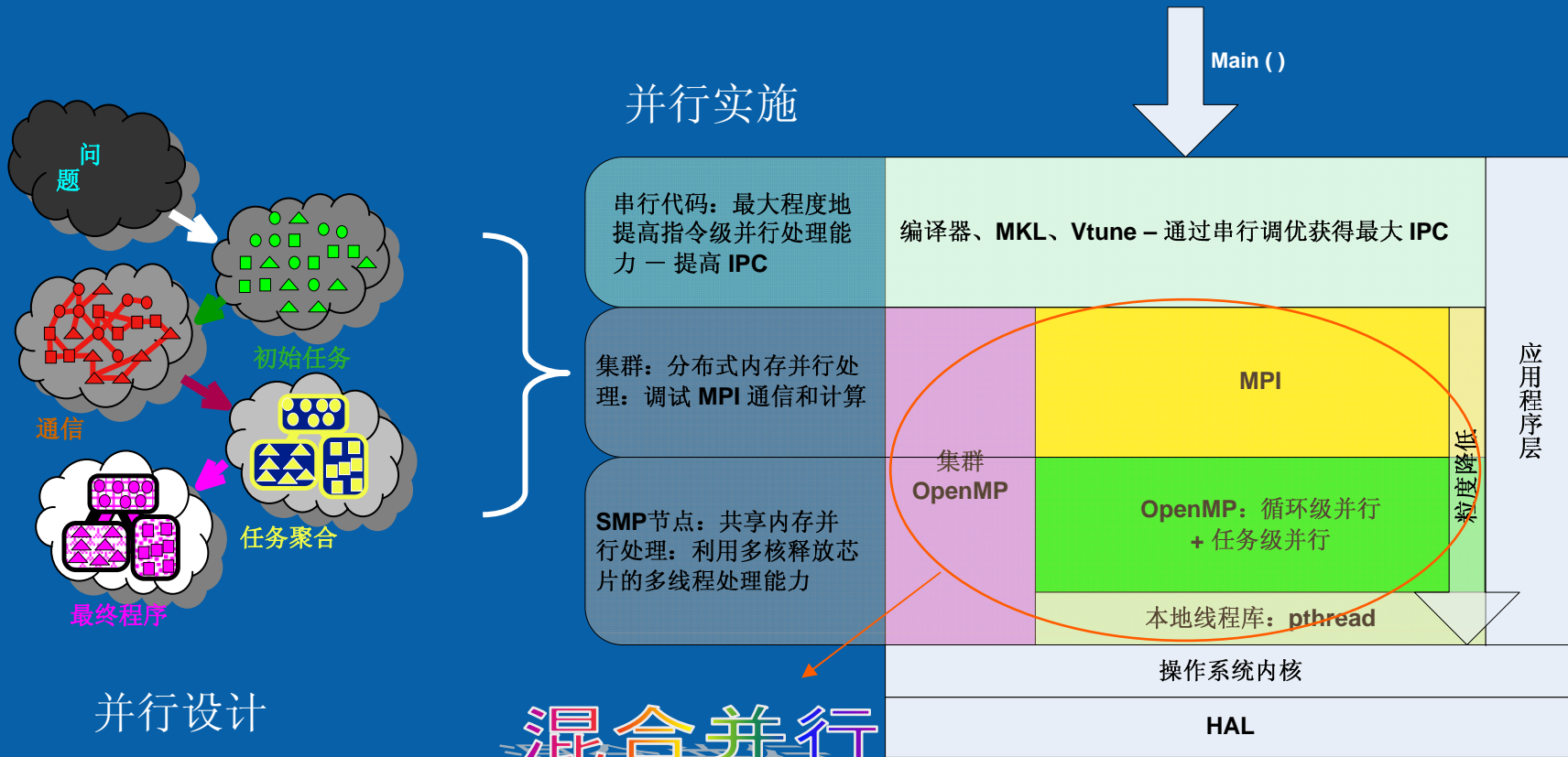
二级

三级



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。



SMP 集群性能模型

- Amdahl 法则的分支:

假定 T_1 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:

$$p \rightarrow 1, \quad \text{Speedup} \rightarrow N$$

- 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, $N \approx N_{\text{MPI}} * N_{\text{thread}} * \text{TR}\%$, 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, $\text{TR}\%$ 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

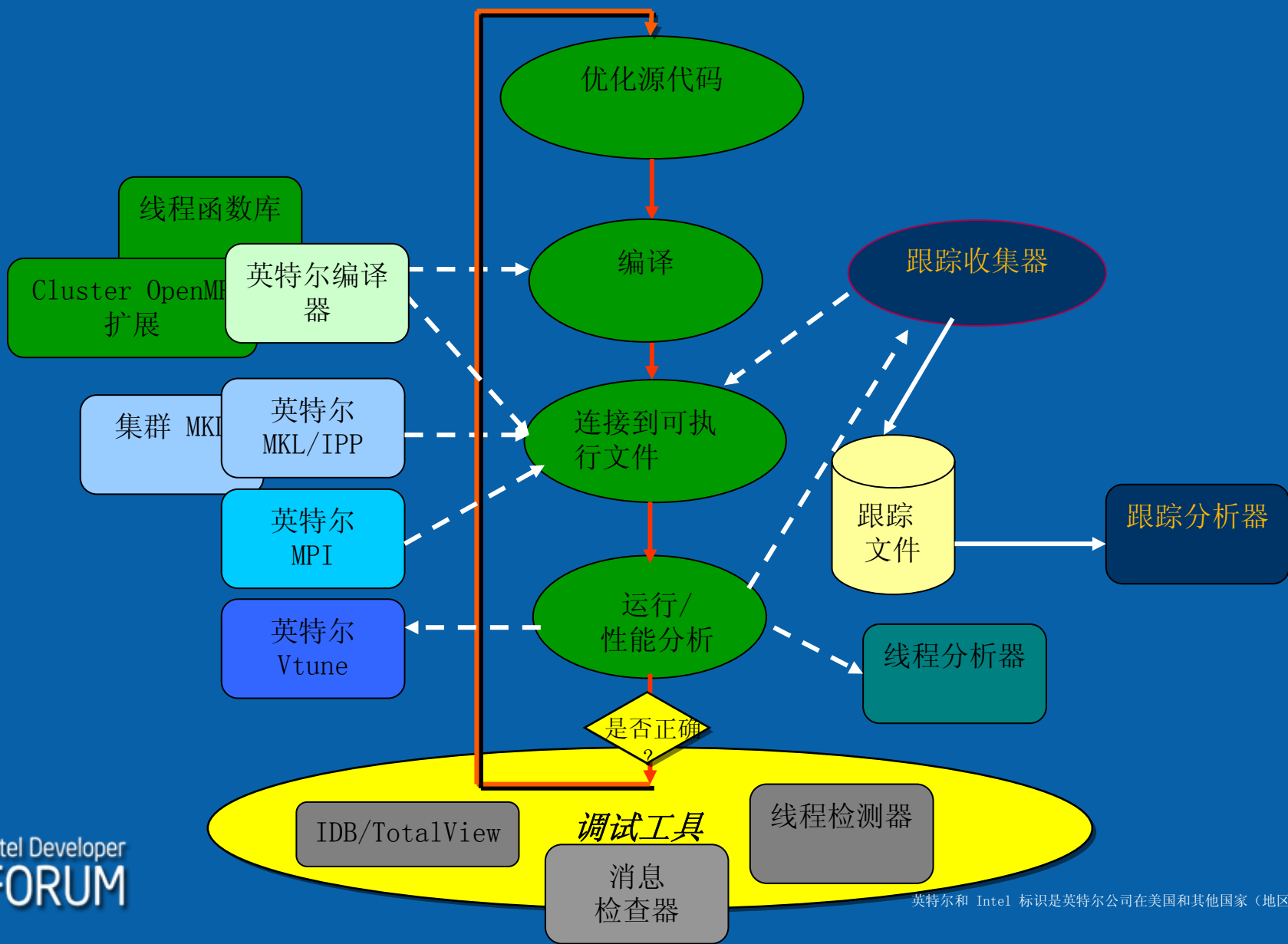
- 如果将多线程处理代码集成到 MPI 模型中，则我们得到

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

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。 $\sum T_{th}$ 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 $T_{mpi} + \sum T_{th} < T_{mpi}$ ，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

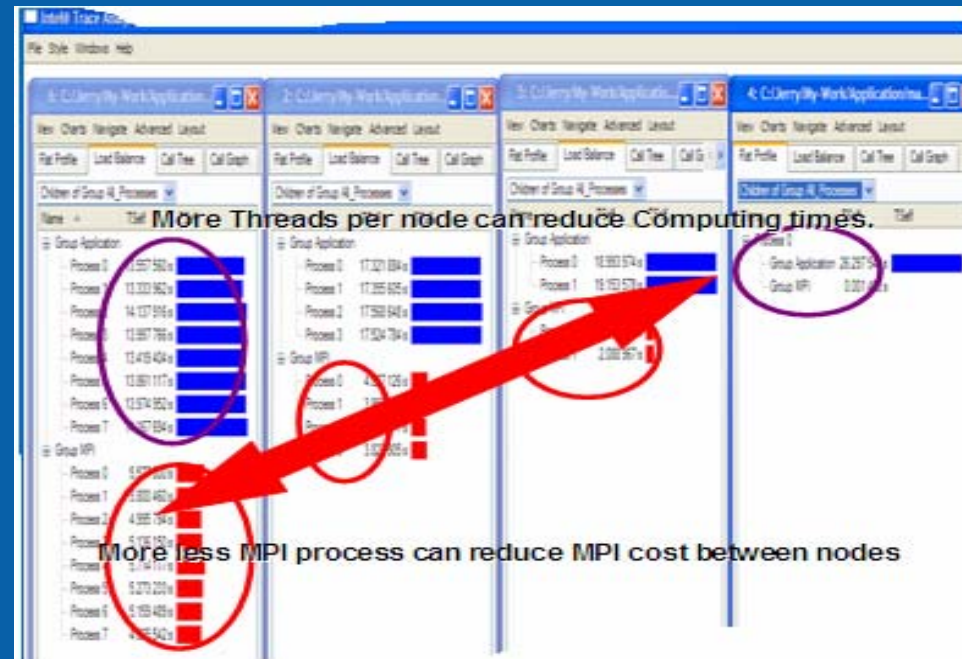
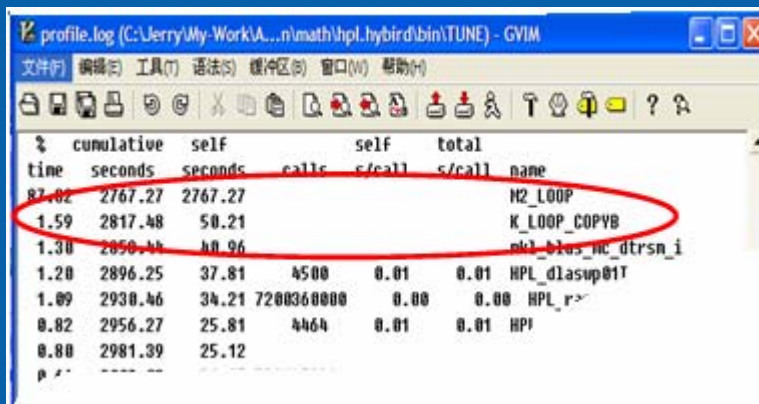
混合编程工具：编译器/ITAC/ITPT



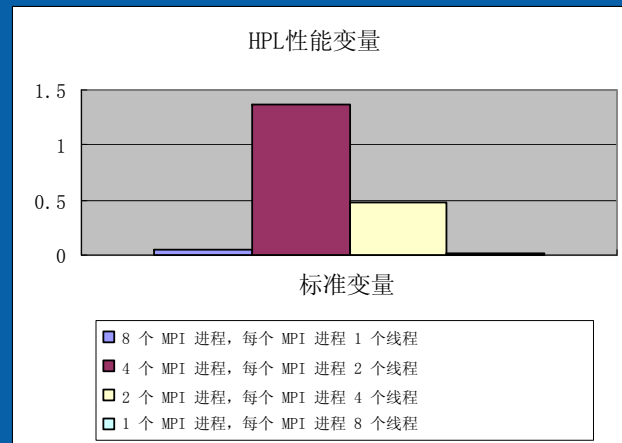
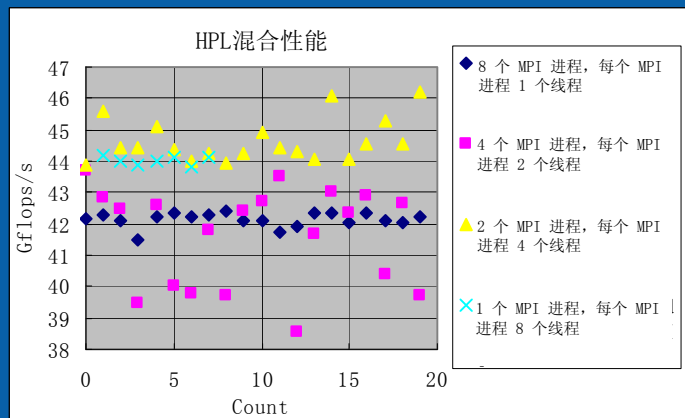
将混合模式应用于 HPL

- 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？

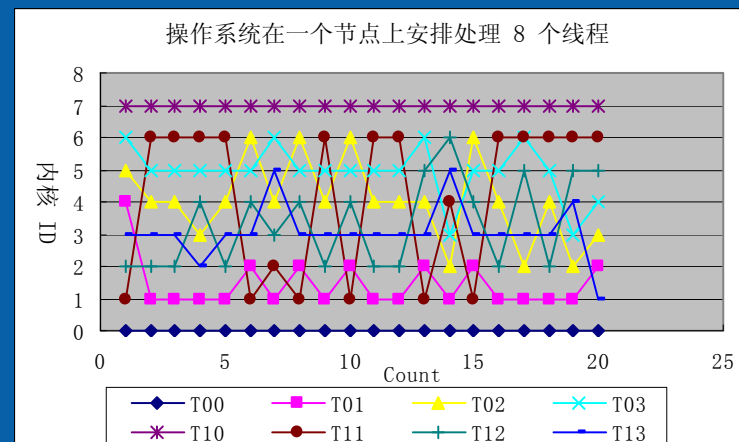
	内核数量 (MPI 进程 x 线程)	性能 (Gflops/s)
MPI 性能	8 x 1	42.48
OpenMP 性能	1 x 8	44.21
混合性能	2 x 4	46.43



由于处理器间的线程漂移 (threads drifting) 导致性能统计波动

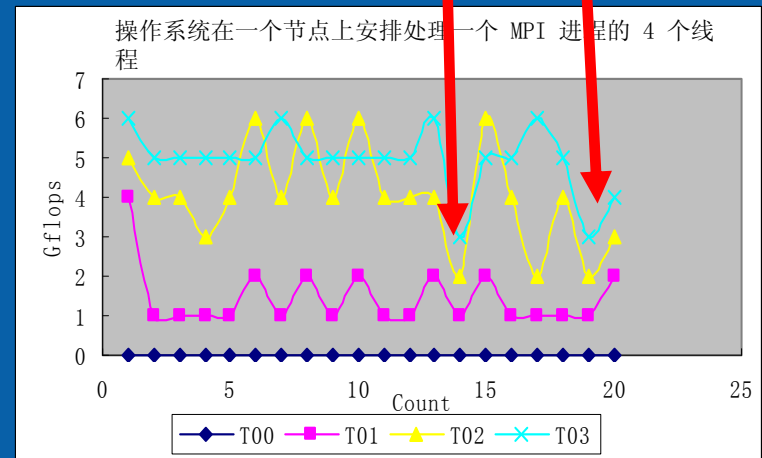
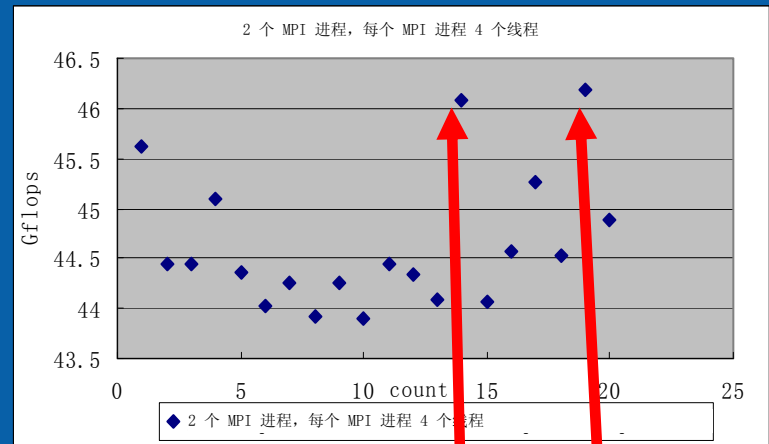


- 在每个线程处理应用程序中，性能均有统计波动。
- 除非将线程绑定到处理器内核，否则波动是不可避免的
- 运行时处理器间存在线程跳跃

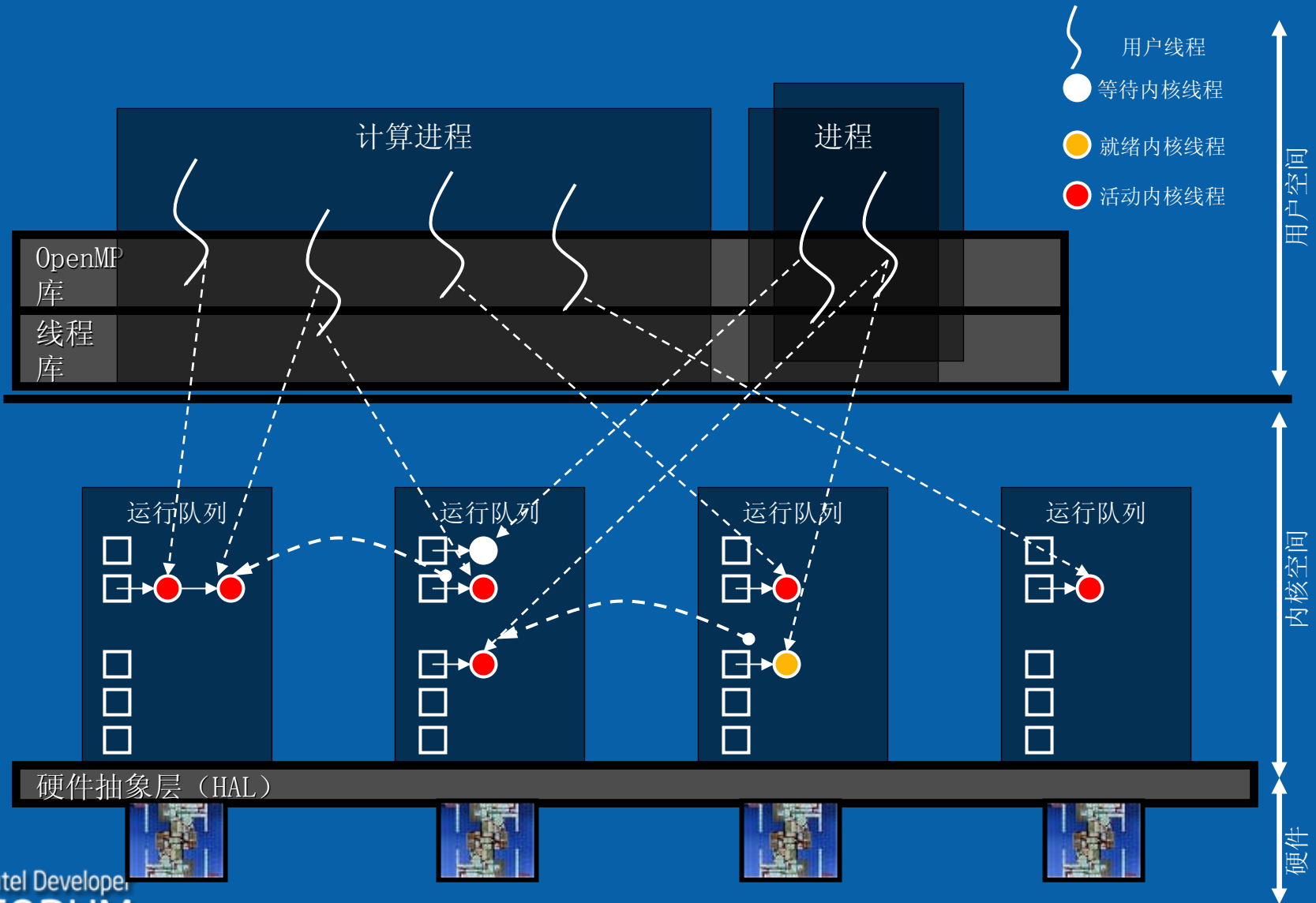


操作系统线程调度检测到的线程偏移

- 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



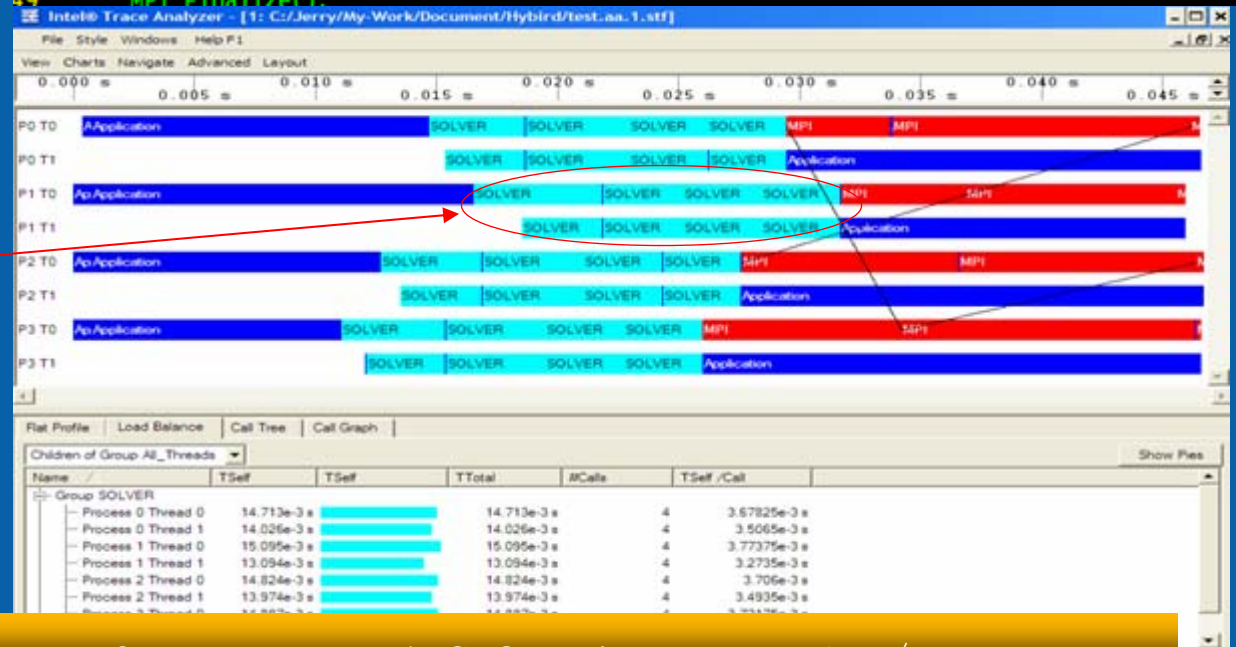
Linux 2.6 中的 OpenMP 线程



ITAC 支持通过手动添加探测代码跟踪 OpenMP 线程

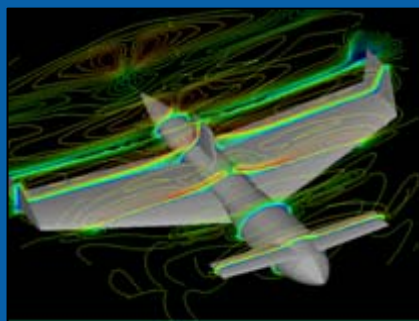
- 第 1 步：在函数的源代码中插入函数 VT_funcdef/VT_enter/VT_leave，然后在线程中执行这些函数
- 第 2 步：编译源代码时链接到英特尔® 跟踪分析器工具库
- 第 3 步：运行应用程序。
- 第 4 步：应用程序运行完成后，通过英特尔® 跟踪分析器工具观察跟踪数据。
- 注意线程处理部分
- 注意插入的探测代码带来的开销

```
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();
```

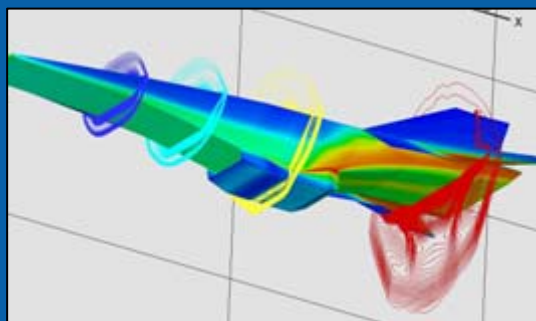


NAPA 混合并行：调试 MPI 和 OpenMP

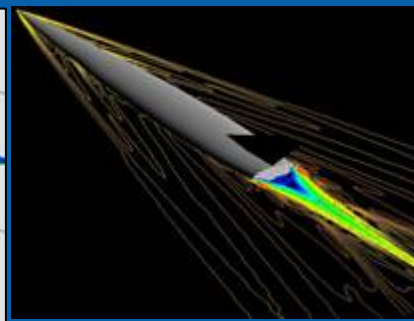
- Napa* 是 NUAU 自己开发的 CFD 应用程序，英特尔已对其代码进行了彻底的 MPI 优化
- MPI 版本有许多点对点通信，可用于在每次迭代时更新影子网格 (ghost grid) 的数据，并且集中通信量不多
- MPI 版本在以太网上的速度提升不到 60，可扩展能力较低。这是应用混合并行模型的一个很好的试验对象



MAV



X-43



NASA 升力式航天器

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)

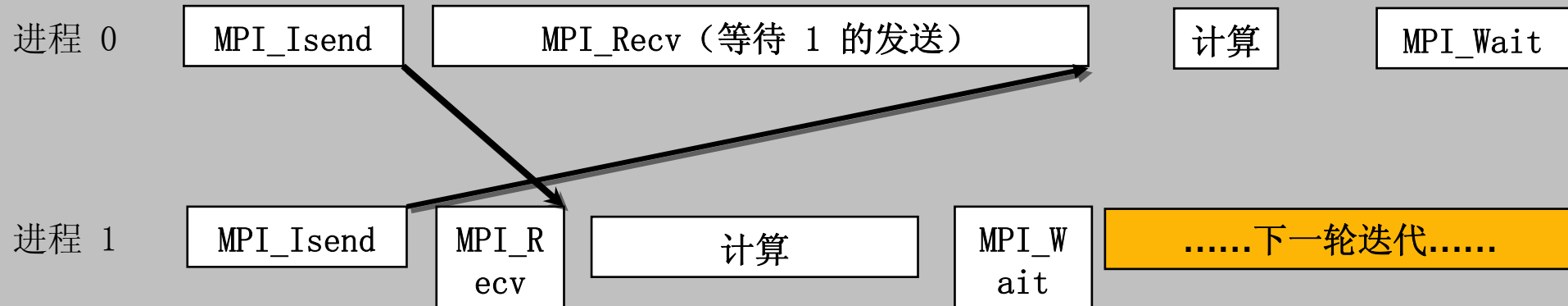
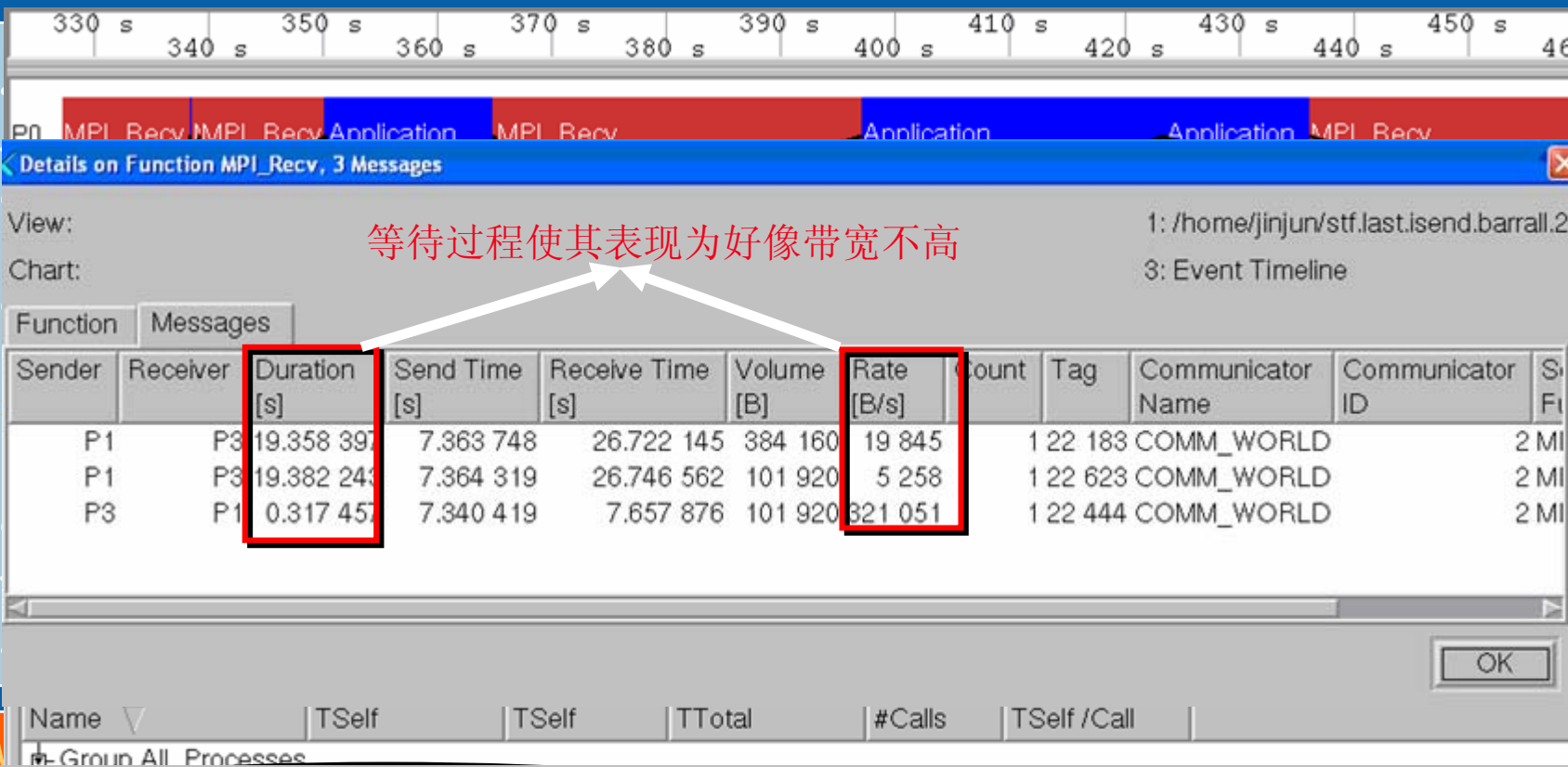


OpenMP 不能够很好地支持从 2 个内核到 4 个内核的扩展，其开销 $T_{th} > T_{mpi}$

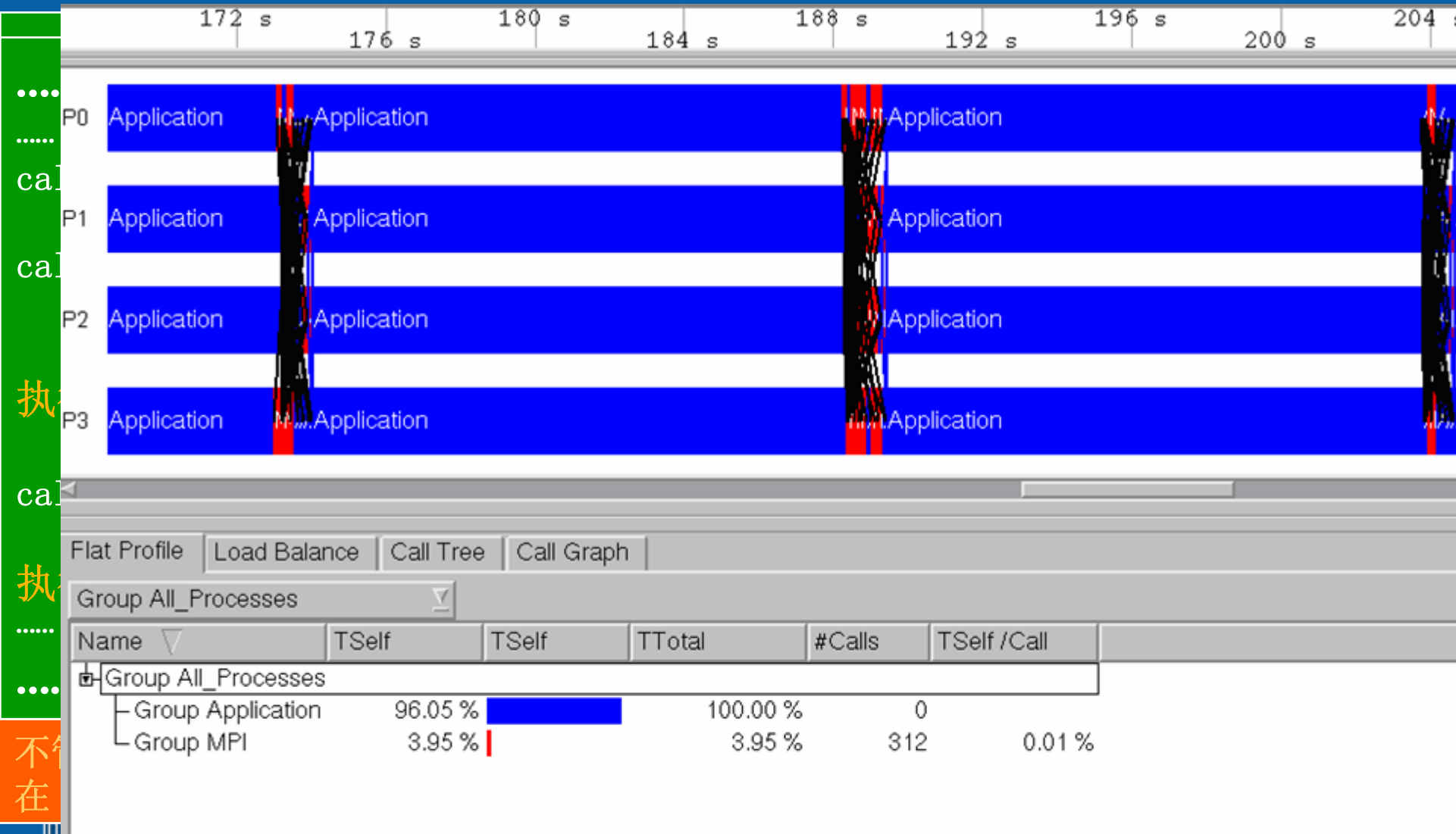
MPI 在 4 个内核时性能更高

NAPA: MPI 通信和计算

计算网格: 97*97*97*24

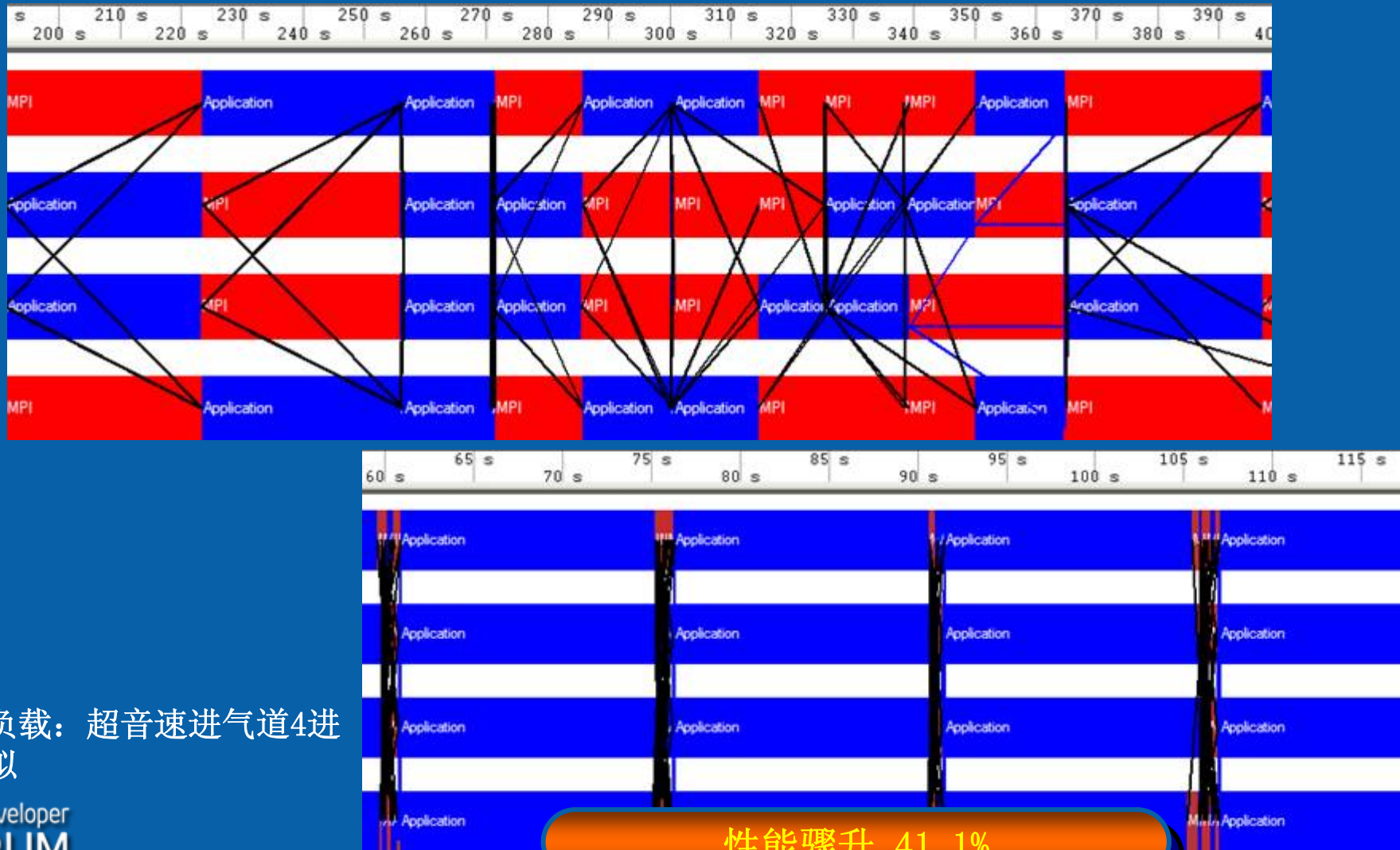


MPI 通信和计算 (续)



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

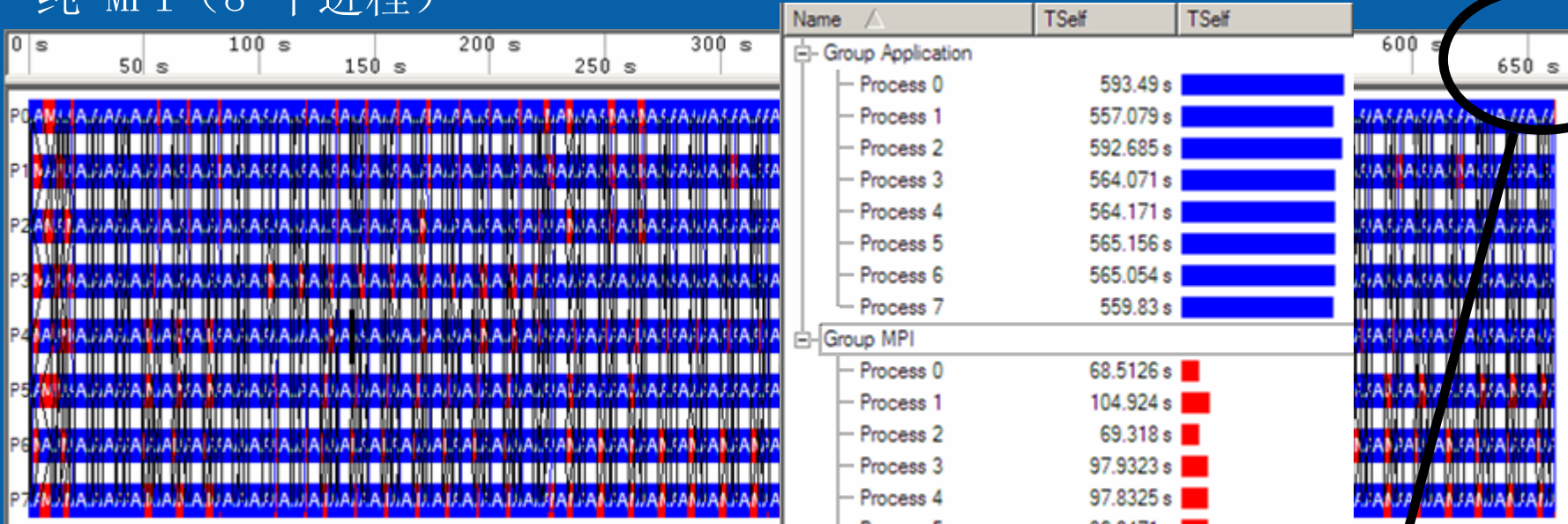
NAPA: MPI 并行优化应用于实际的工作负载



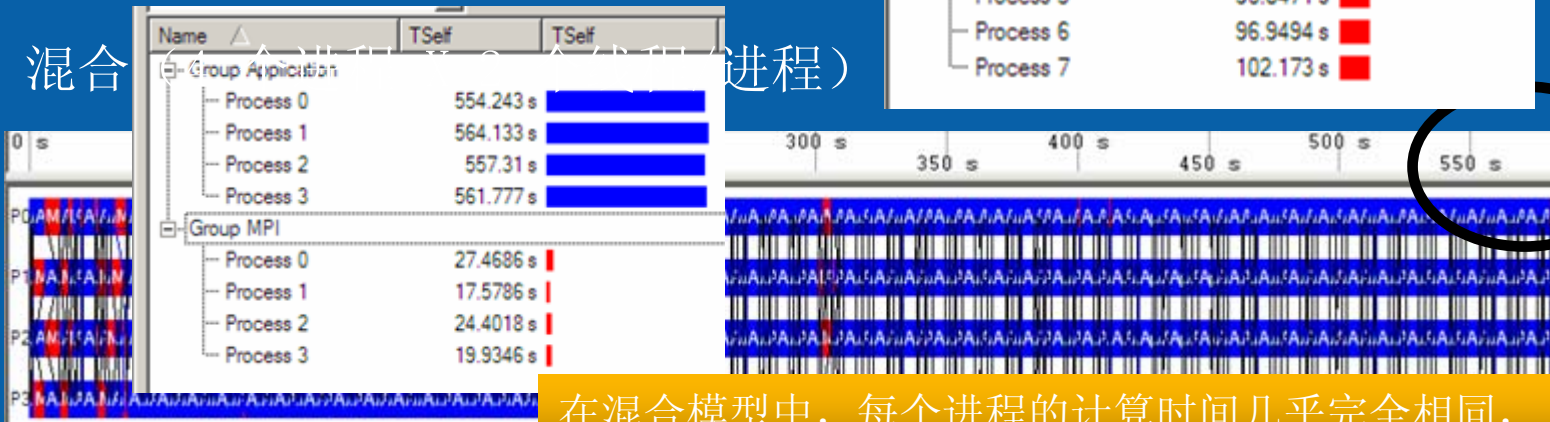
工作负载：超音速进气道4进程模拟

线程减少了计算量

纯 MPI (8 个进程)

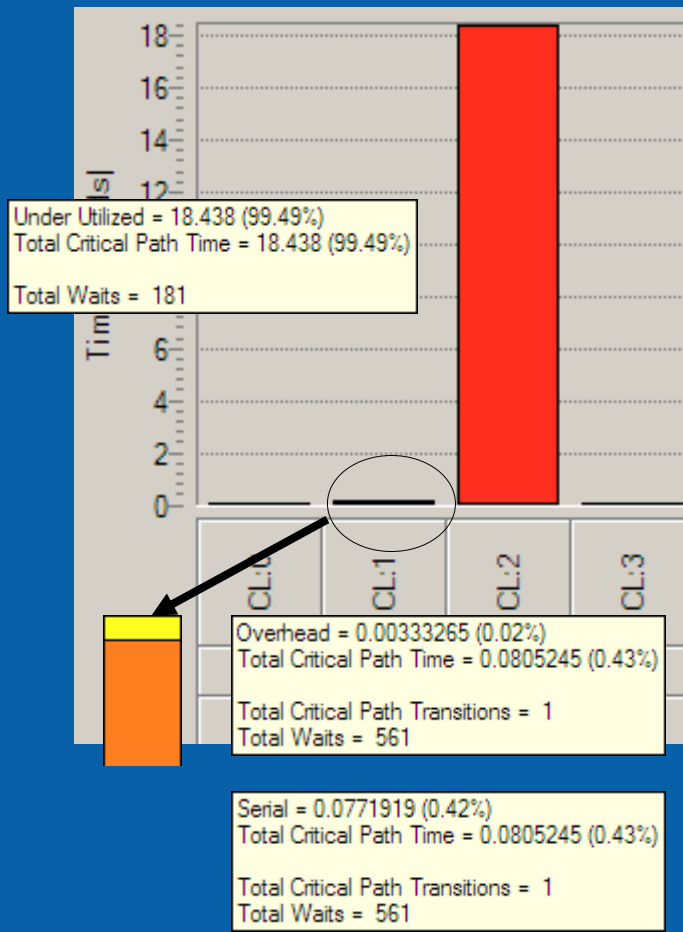


混合

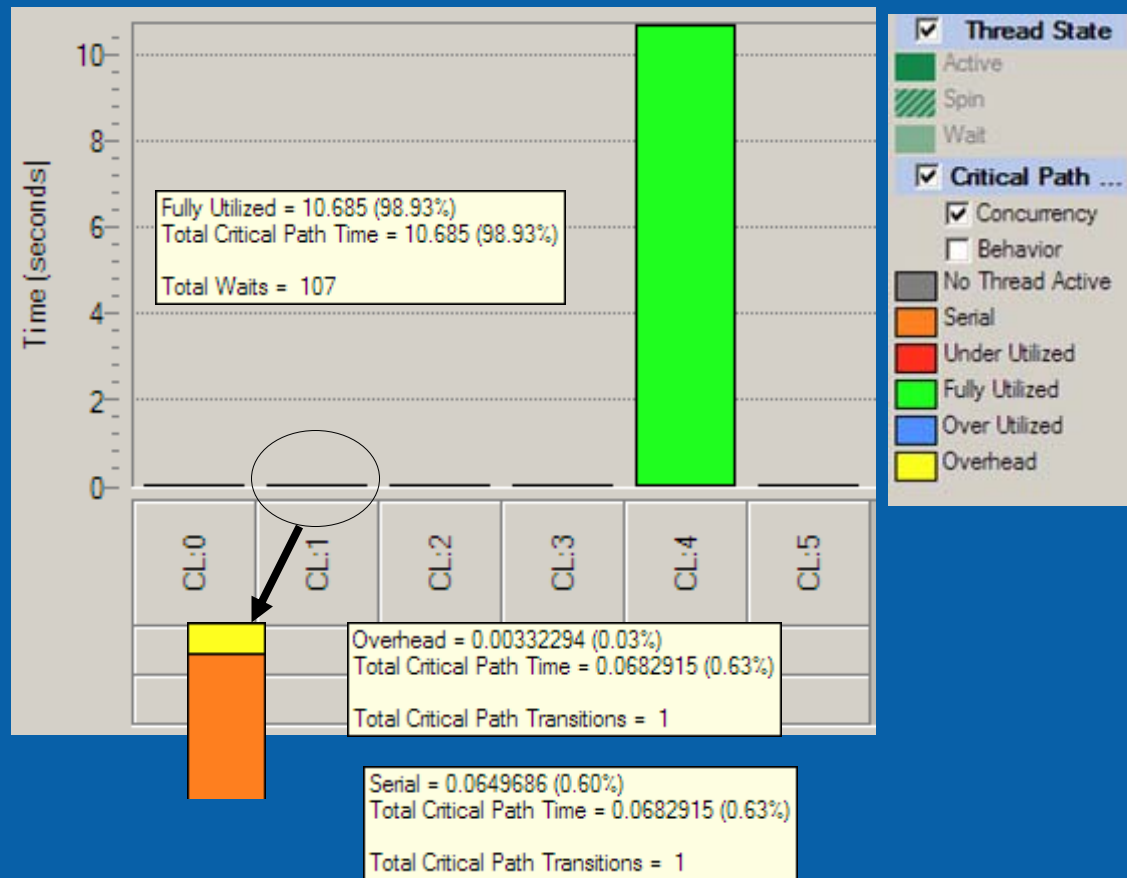


在混合模型中，每个进程的计算时间几乎完全相同，通信量大幅减少，从 13.9% (纯 MPI) 减少到 3.84% (混合)

通过英特尔线程调节器调节 OpenMP 线程



2 个线程



4 个线程

进行 MPI 调试后，避免简单化OpenMP - 对其进行优化

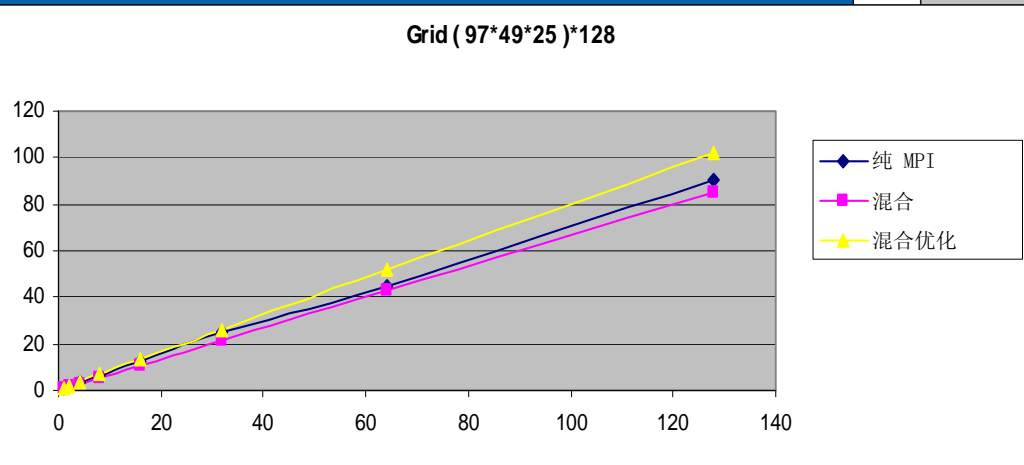
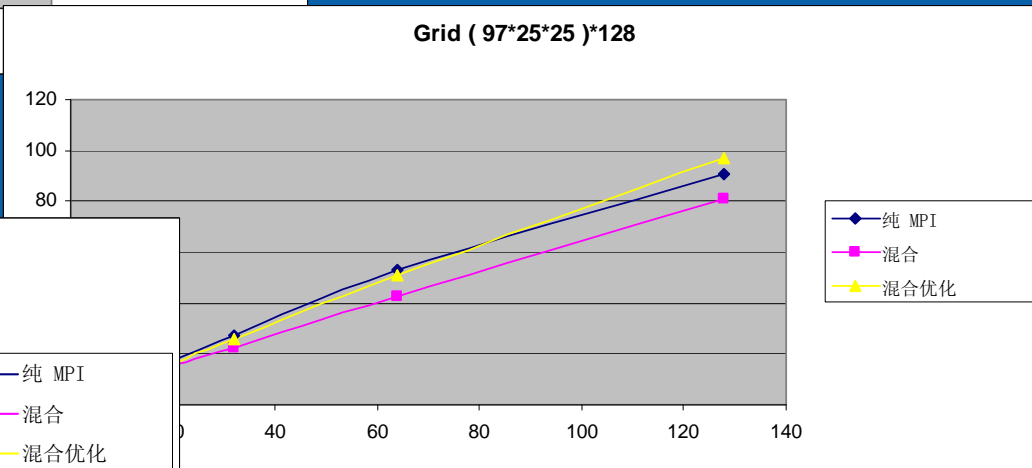
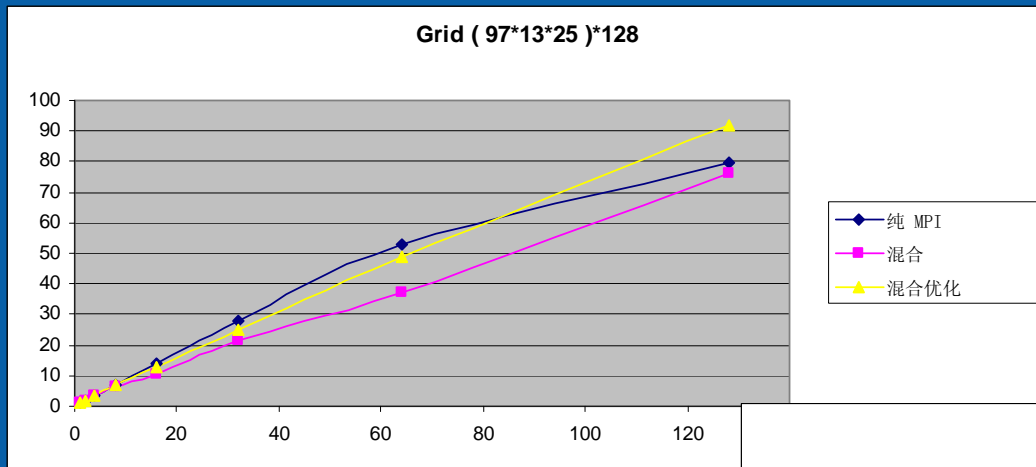
```
1
2 C-----
3 C      NEXT PLANES
4 C-----
5 DO 1300 I = 3, IL-1
6
7     DO 1200 K = 2, KL
8     DO 1200 J = 2, JL
9     AM(J,K) = AD(J,K)
10    AD(J,K) = AP(J,K)
11
12
13     DISI = DISS(I+1, J, K, 1)
14     CFLI = CFL*DISI/(RADI(I+1, J, K)+MAX(RADJ(I+1, J, K), RADK(I+1, J, K)))
15     CFLA = 0.25 * (CFLI*CFLI/(CFL*CFLL)-1.)
16     AP(J, K) = MAX(CFLA, SMOOP1)
17
18
19
20
21
22
23
24
25 1200 CONTINUE
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

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

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

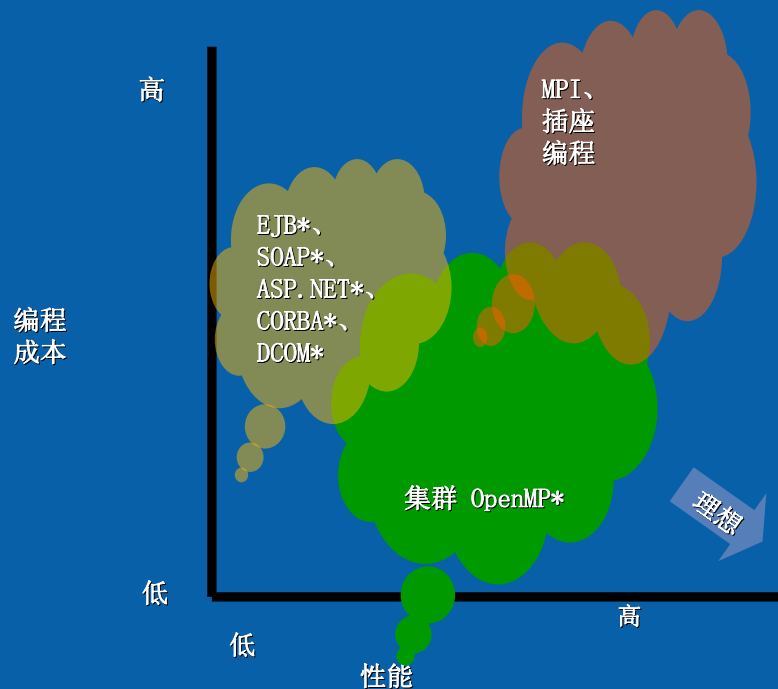


- 对于混合模型，每个进程包含两个线程
- MPI 性能指标评测在相同的节点时实现了共享存储

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

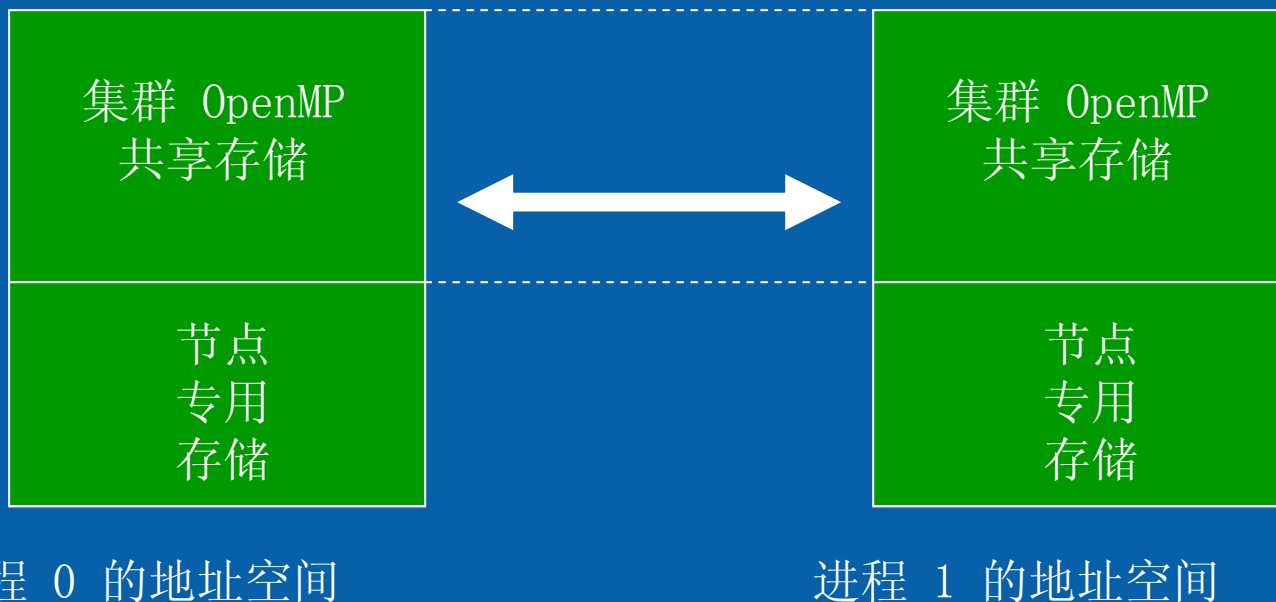
- 英特尔Cluster OpenMP 是一种 OpenMP 扩展，可跨集群节点部署 OpenMP 线程。
- Cluster OpenMP* 是一种较易实施的集群编程方式。有些代码可通过集群 OpenMP 获得良好的性能和扩展。将代码移植到 Cluster OpenMP 是一种系统化过程，比将进程移植到消息传递的工作量要少得多。可以使用性能工具帮助优化程序，在集群中获得最佳性能。

支持在英特尔架构 64 位集群上执行 OpenMP 应用程序。



集群 OpenMP 工作原理

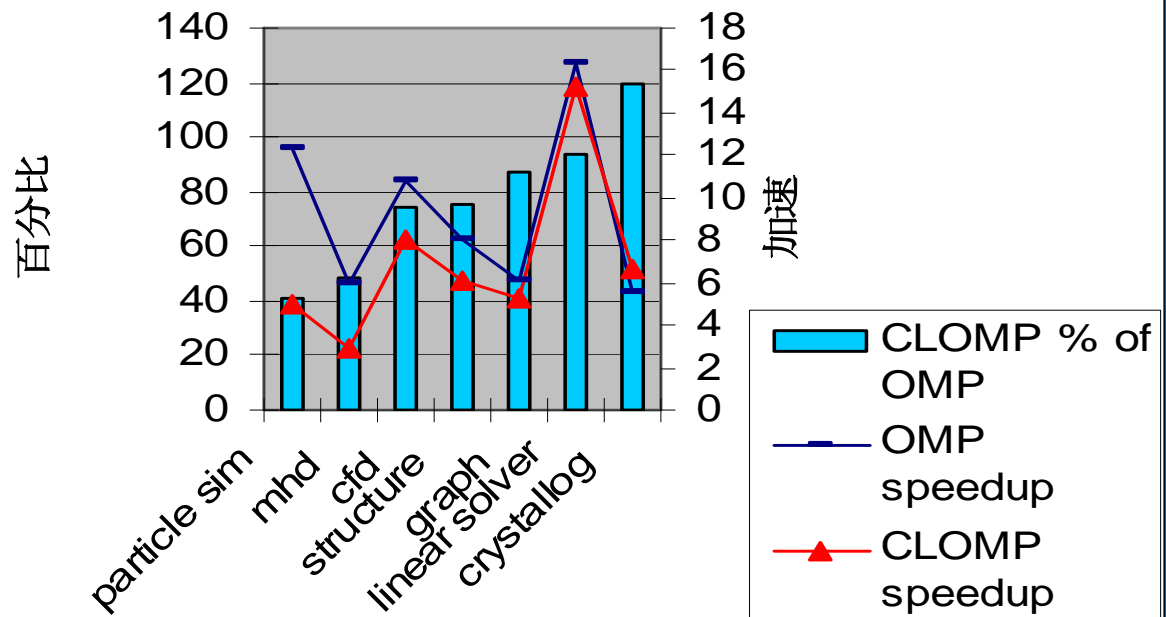
集群 OpenMP* 存储模型



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

集群 OpenMP* 相比 OpenMP 的加速比 16 个进程 (Gig E)

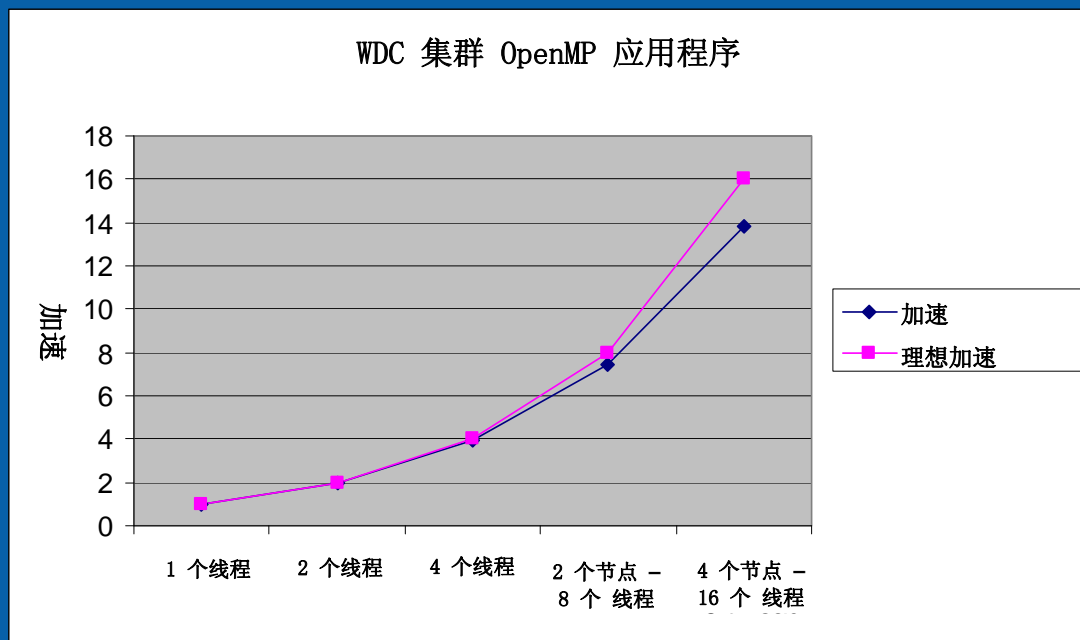


*文中涉及的其他名称及商标属于各自所有者资产。

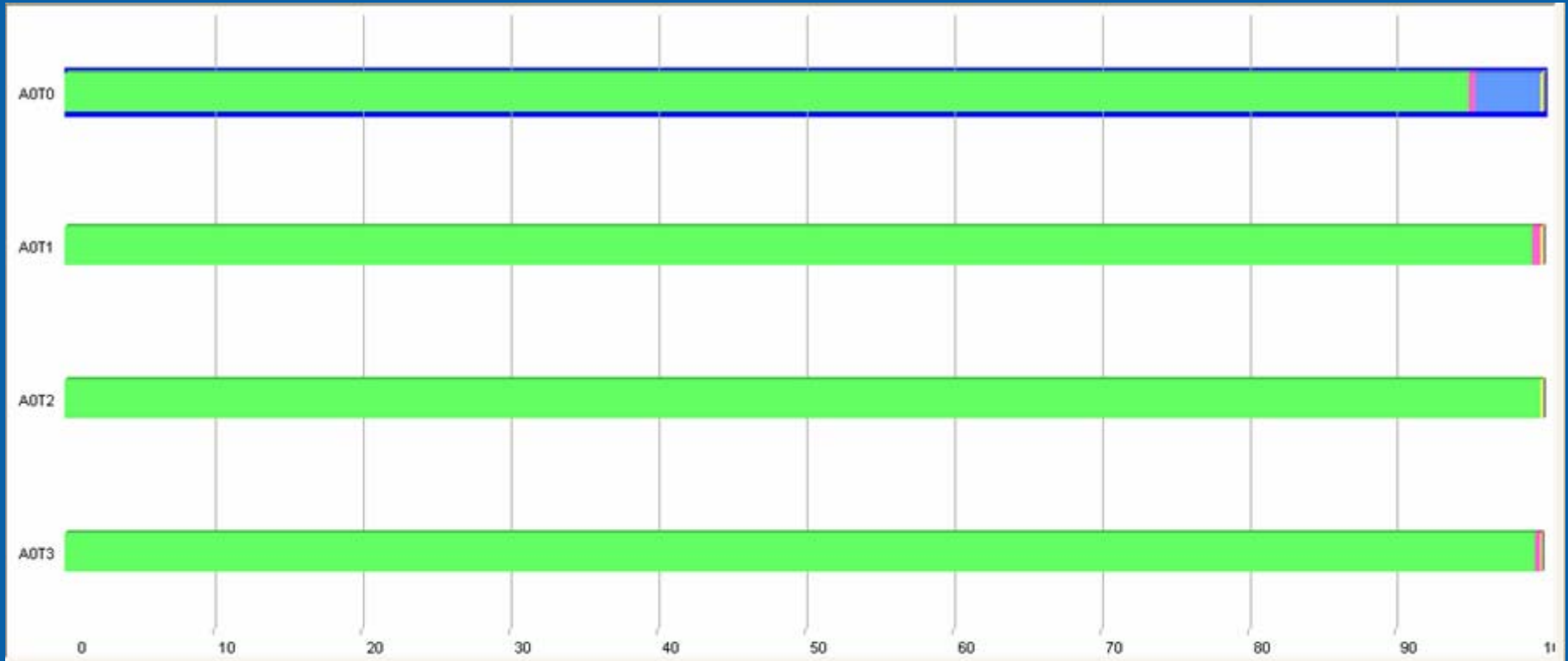
案例研究： 双路至强 5100 混合集群中的 GalRen

- 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



Label	Total	% Parallel	% Sequential	% Imbalance	% Barrier	% Locks	% Synchronized	% Parallel overheads	% Sequential overheads
A0T2	11	99.815	0.000	0.000	0.085	0.000	0.000	0.100	0.000
A0T0	11	95.026	4.471	0.000	0.407	0.000	0.000	0.094	0.000
A0T1	11	99.305	0.000	0.000	0.596	0.000	0.000	0.099	0.000
A0T3	11	99.476	0.000	0.000	0.426	0.000	0.000	0.098	0.000

95% 以上为并行

4% 为串行

0.5% 用于障碍和同步

2 个节点中有 8 个线程



结论：
调试方向： 减少串行部分

Label	Total	% Parallel	% Sequential	% Imbalance	% Barrier	% Locks	% Synchronized	% Parallel overheads	% Sequential overheads
A0T0	8.470	81.418	16.152	0.000	2.257	0.000	0.000	0.174	0.000
A0T3	7.050	97.515	0.000	0.000	2.270	0.000	0.000	0.216	0.000
A0T1	7.060	97.392	0.000	0.000	2.408	0.000	0.000	0.200	0.000
A0T4	7.030	97.835	0.000	0.000	1.991	0.000	0.000	0.173	0.000
A0T2	7.060	97.518	0.000	0.000	2.266	0.000	0.000	0.215	0.000
A0T6	7.030	98.404	0.000	0.000	1.422	0.000	0.000	0.174	0.000
A0T5	7.030	98.543	0.000	0.000	1.280	0.000	0.000	0.177	0.000
A0T7	7.050	98.265	0.000	0.000	1.560	0.000	0.000	0.175	0.000

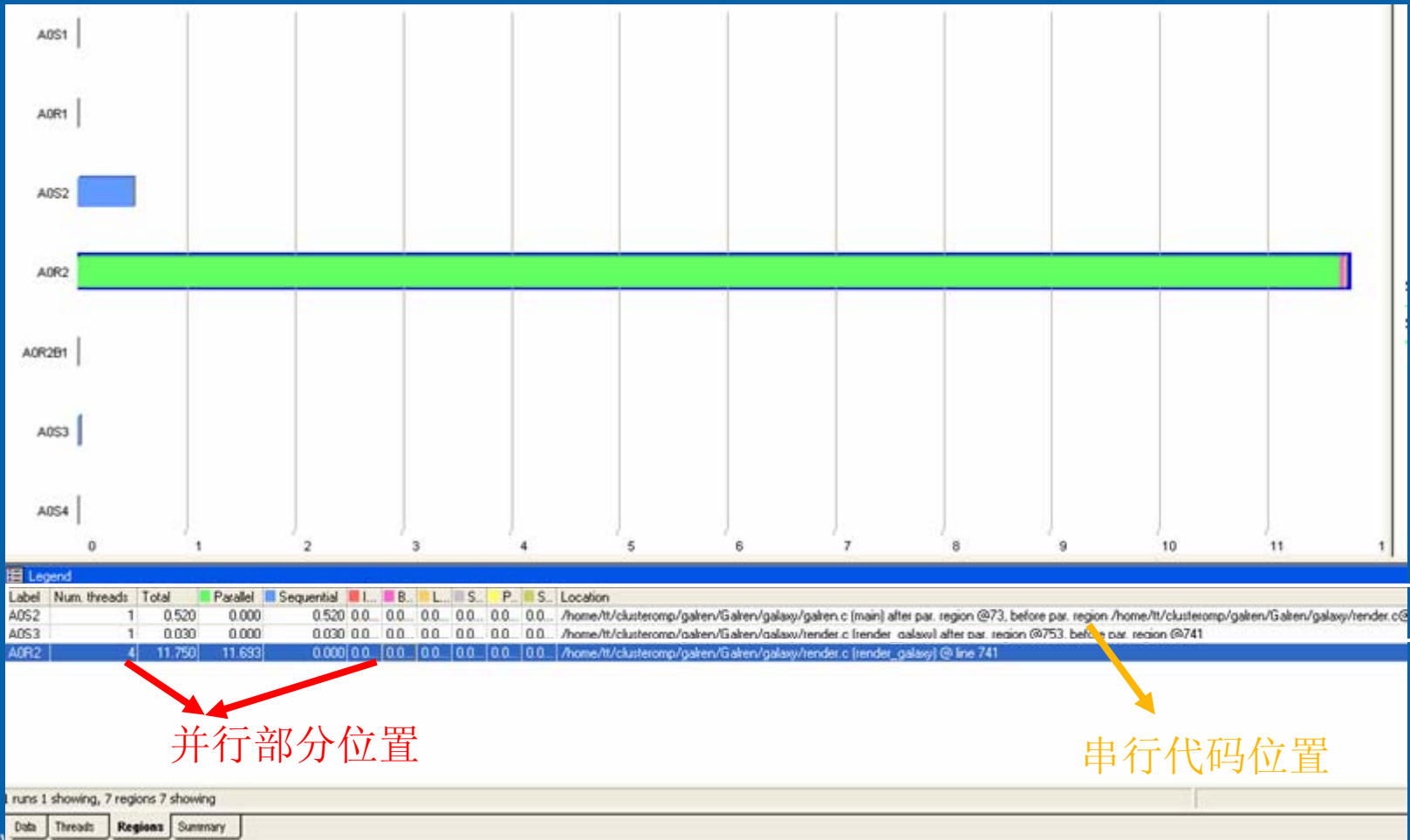
81% 以上为并行

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



并行部分位置

串行代码位置

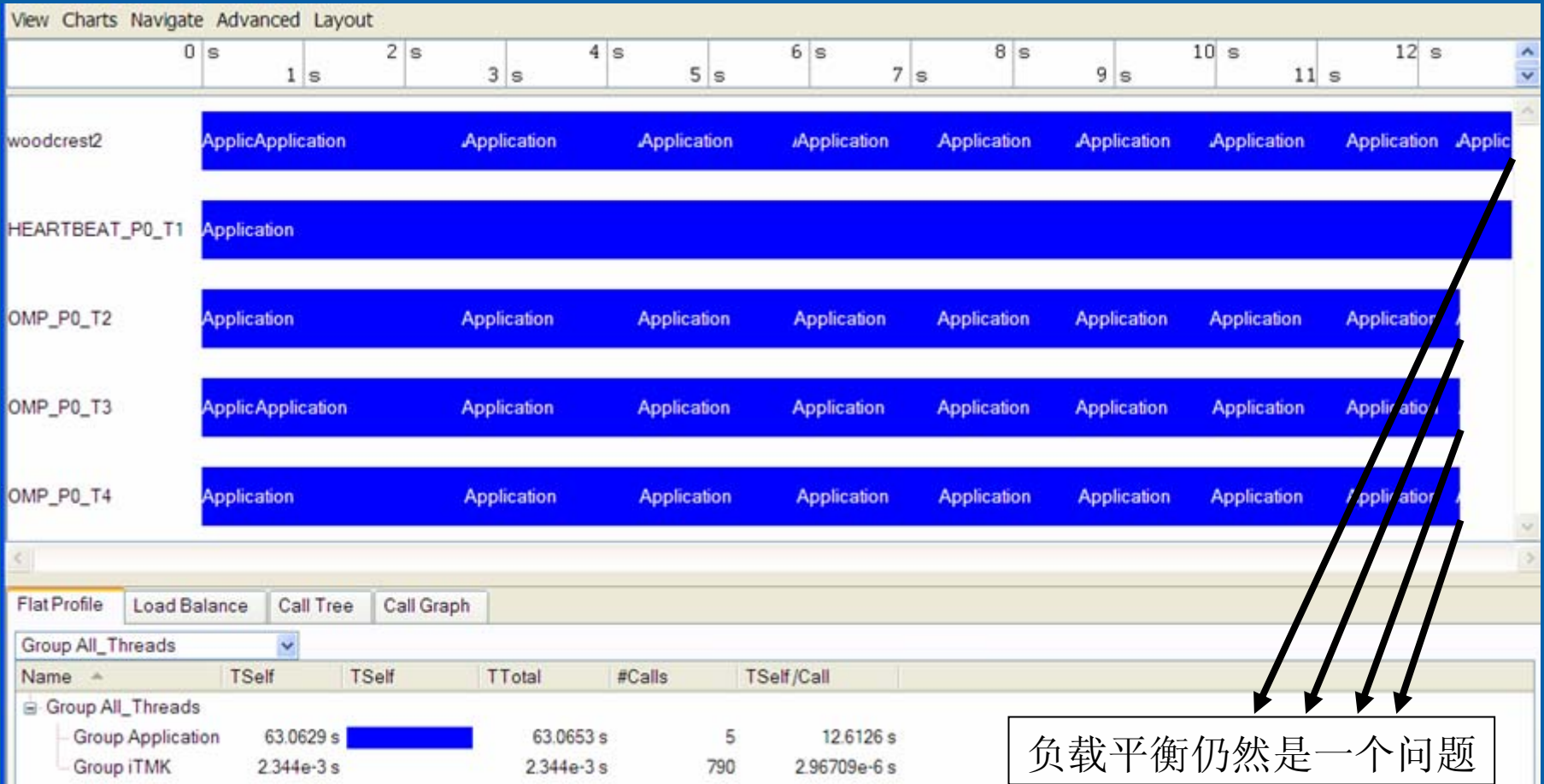
```
#pragma omp parallel private(i, j, R, G, B, A)
{
#if defined(OMP)
    int thread_num = omp_get_thread_num();
    int my_slice = xsize / omp_get_num_threads();
    int my_offset = (thread_num * my_slice);
    int my_bytes = ( sizeof( uByte8 ) * my_slice );
    uByte8 *my_pixels;

    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)



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

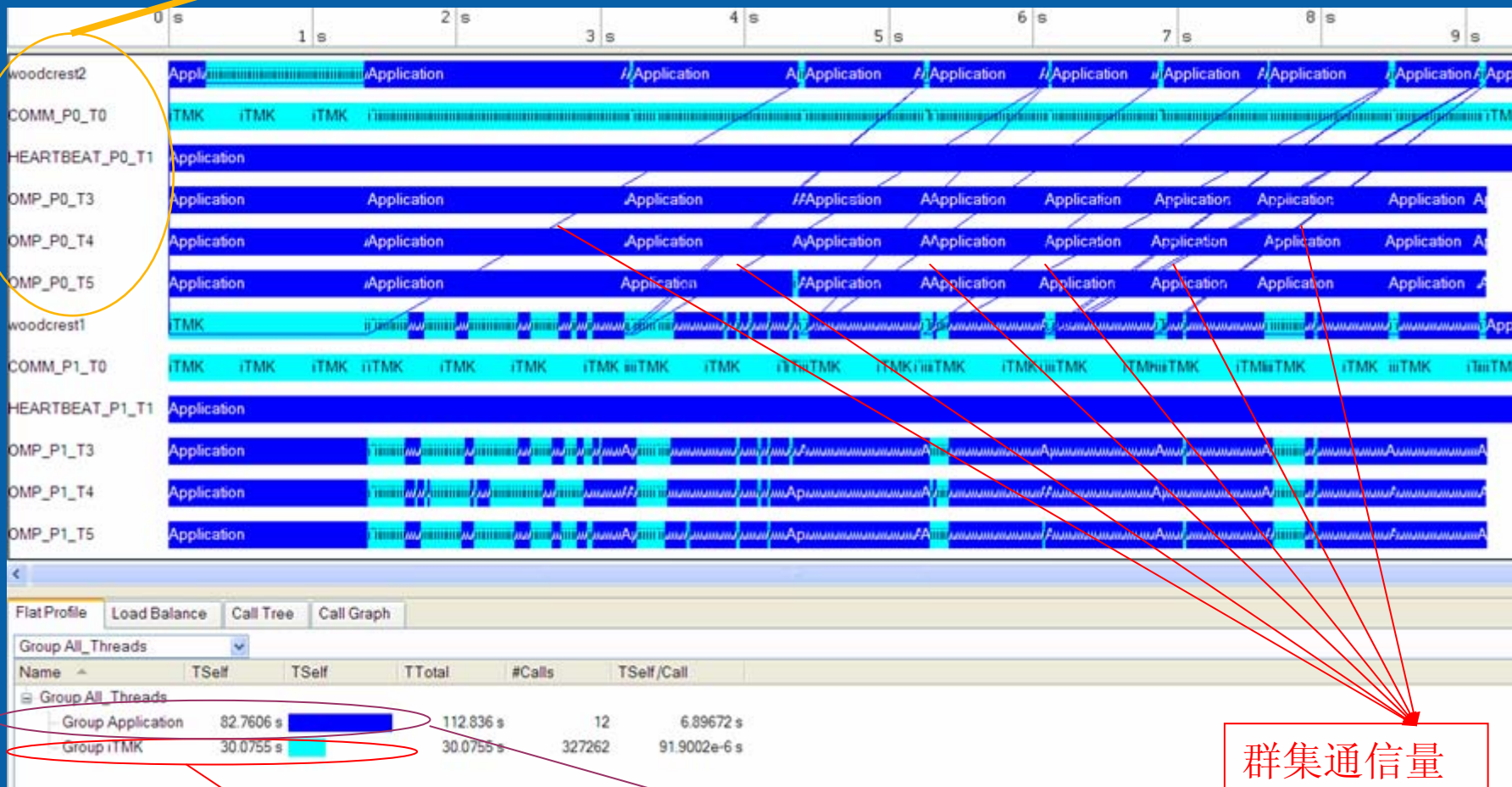


主线程 和
一个节点中用于计算的其
它 3 个次级线程

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



群集通信量

通信部分

计算部分

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

谢谢！

- 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:

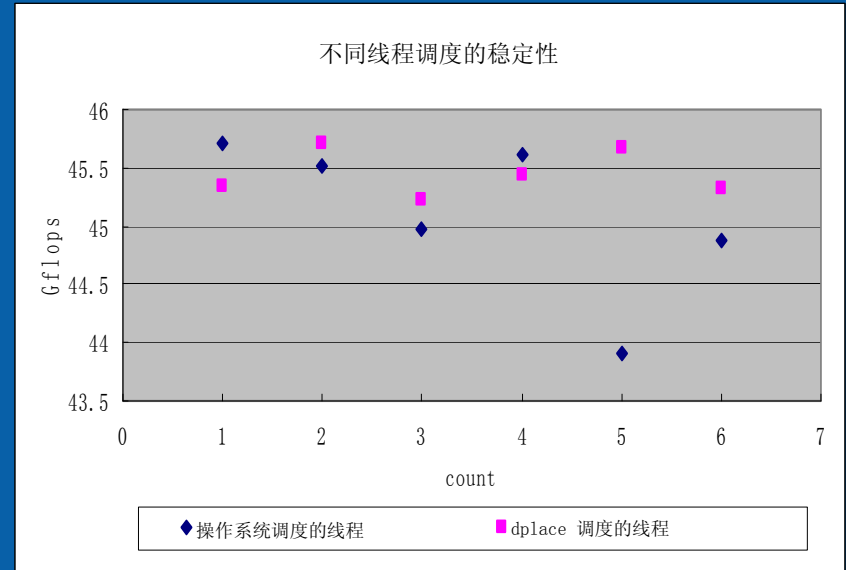
有些操作系统可以提供一些工具，允许在不同的内核|CPU 上调度线程|进程，例如 SGI 或 BULL 的 NUMA 工具。

方法 2:

在 OpenMP 区域中插入 OpenMP 函数 `sched_setaffinity`，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