
intel
Leap ahead™
Intel Developer
FORUM

Hybrid Parallel Performance-Tuning for Multicore-based HPC Clusters

Wanqing He
SSG CRT\HPC Enabling Manager

SSGS004

Intel Developer FORUM

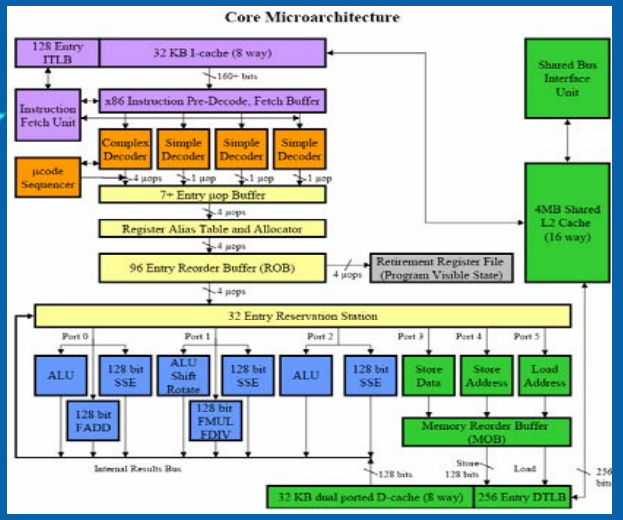
Disclaimer

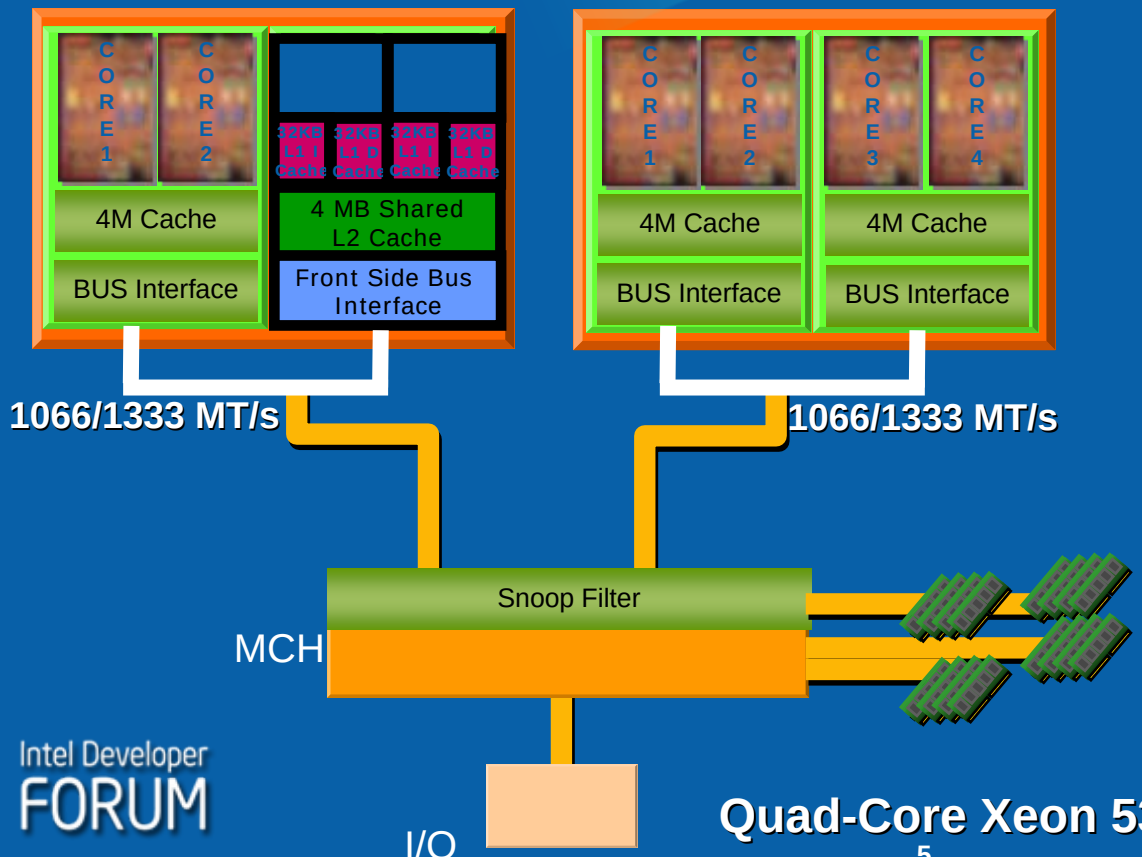
- INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY RELATING TO SALE AND/OR USE OF INTEL PRODUCTS, INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT, OR OTHER INTELLECTUAL PROPERTY RIGHT.
- Intel may make changes to specifications, product descriptions, and plans at any time, without notice.
- All dates provided are subject to change without notice.
- Intel and Itanium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries (regions).
- *Other names and brands may be claimed as the property of others.
-

Agenda

- Define the hybrid parallel performance Tuning model
- Apply hybrid tuning to HPL: understand the threads dithering by ITAC
- Tune both MPI and OpenMP: apply hybrid tuning to real-world CFD application
- Try Cluster-OpenMP to fast deploy your Cluster app.
- Summary/Call to action

Intel's Xeon 5300 build SMP cluster base on Core's performance/watt



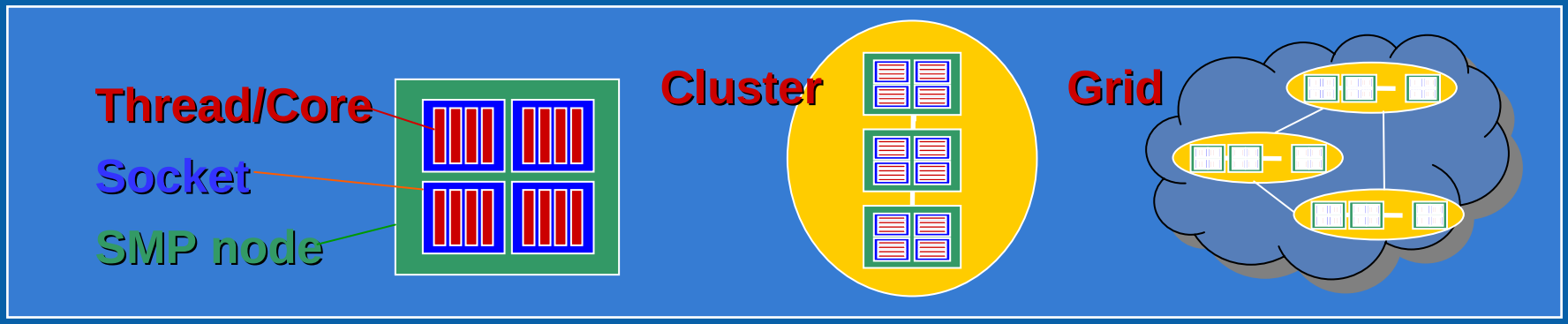


Hybrid parallel tuning: Defined as MPI tuning between SMP nodes mixed with multithreading tuning inside SMP nodes to get both distributed mem. parallelism and share mem. parallelism.

Quad-Core Xeon 5300

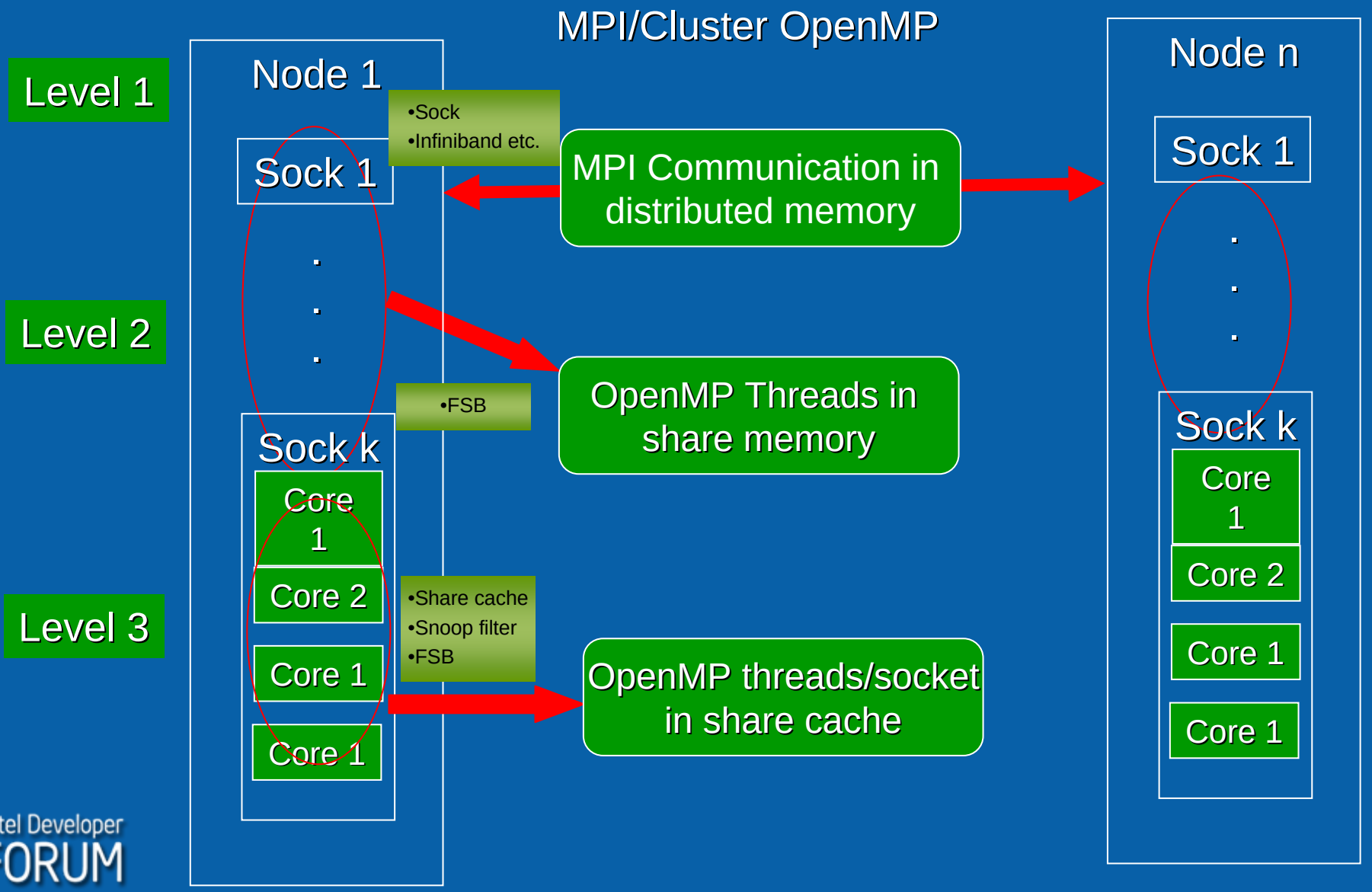
Intel Developer FORUM

Multicore building block for HPC



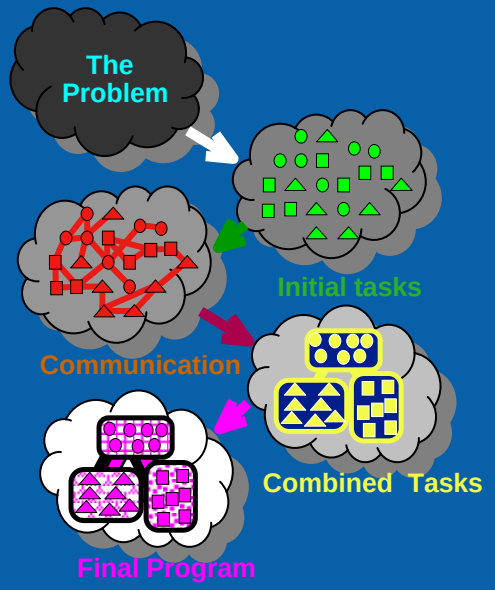
HPC System	Share memory/node	Distributed Memory /Hybrid Cluster	Compute Grid /
Program Model	Multithreading in share mem: OpenMP, pthread, winthread	MPI + OpenMP In distributed mem: Intel MPI, ClusterOpenMP	SOA, Virtualization
Develop Tools	*Intel® Thread Checker* *Intel® Thread Profiler* *VTune®* *Intel® MKL*	*Intel® MPI* *Intel® Trace Collector/Trace Analyser* *Intel® Cluster MKL* *Intel Message Checker* *Intel Threading tools*	*UNICORE, Globus*, etc.* *DRMAA** *Grid Programming Environment (GPE)*

Multicore-SMPs Cluster Model



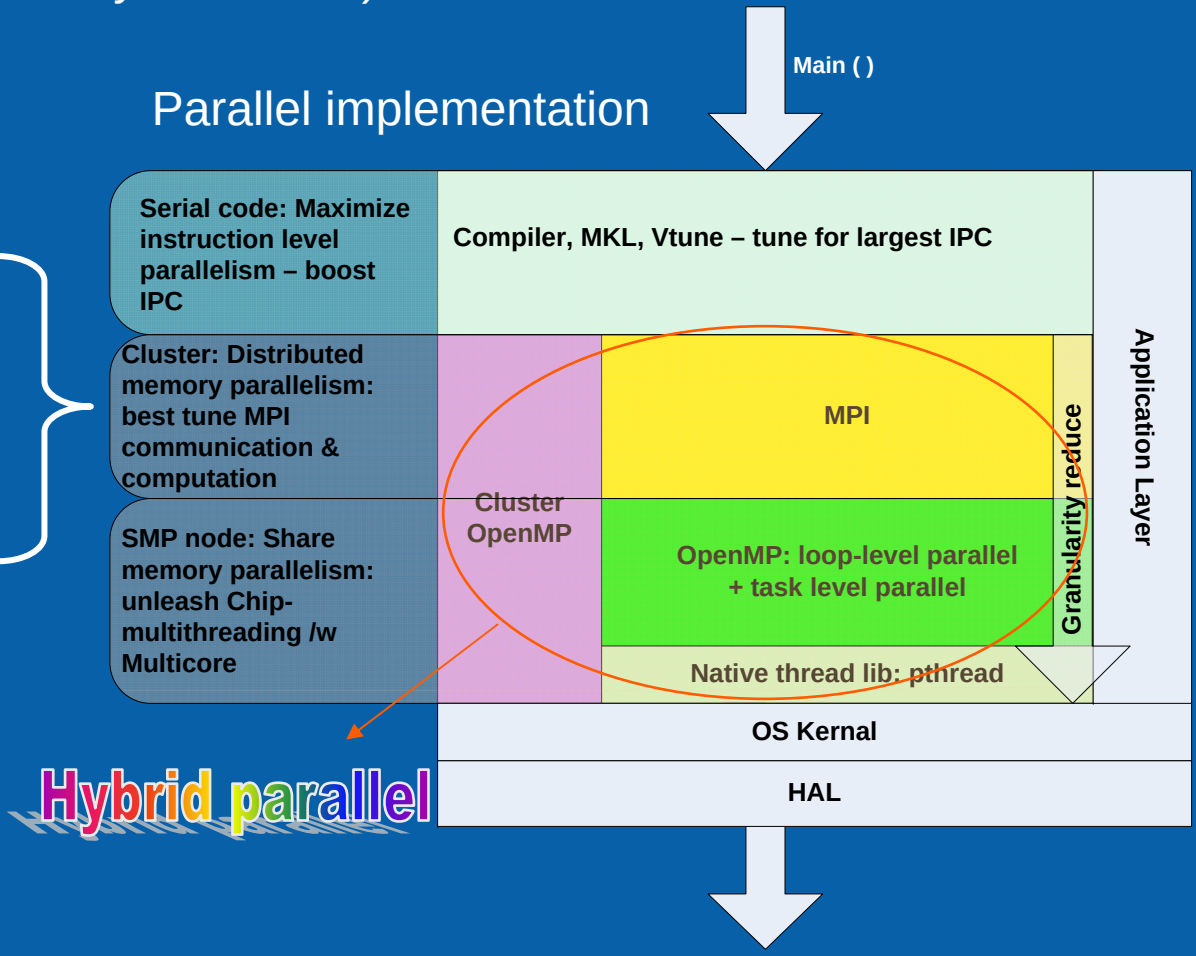
SMP Cluster parallel Model

Hybrid parallel should be implemented from coarse to fine granularity from distributed memory (MPI), share-memory (OpenMP/threading) and instruction level (serial code by IPC boost).



Parallel design

Parallel implementation



SMP Cluster Performance model

- Stem from Amdahl law:

Let T1 be the execution time in serial mode; If p is the fraction of the code that is parallel, then:

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

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

- As we scale the problem for large N, the serial work remains fixed, while the parallelizable work grows with N.

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

- *It is possible to realize linear speedup, if we also grow the problem size (Gustafson's law)*
- In hybrid mode, $N \approx N_{MPI} * N_{thread} * TR\%$ where N_{MPI} is the number of MPI processes and N_{thread} is the number of parallel threads running inside one MPI process, and TR% is the threading ratio that represent the percentage of Multi-threaded work in one process

Real world Cluster perf. Model : MPI

- Considering overhead time from MPI, we have:

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

  When considering MPI Model, the communication effects limit the absolute performance and scalability:
  - Latency or message initiation time (T_L)
  - Bandwidth, a measure of transfer rate once it is initiated (B_w)
  - Collective communication calls
  - Synchronization, load imbalance

  Let M = average message size

- T_c = collective communication time
- T_i = Load imbalance or synch time

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

Real world Cluster perf. Model : hybrid

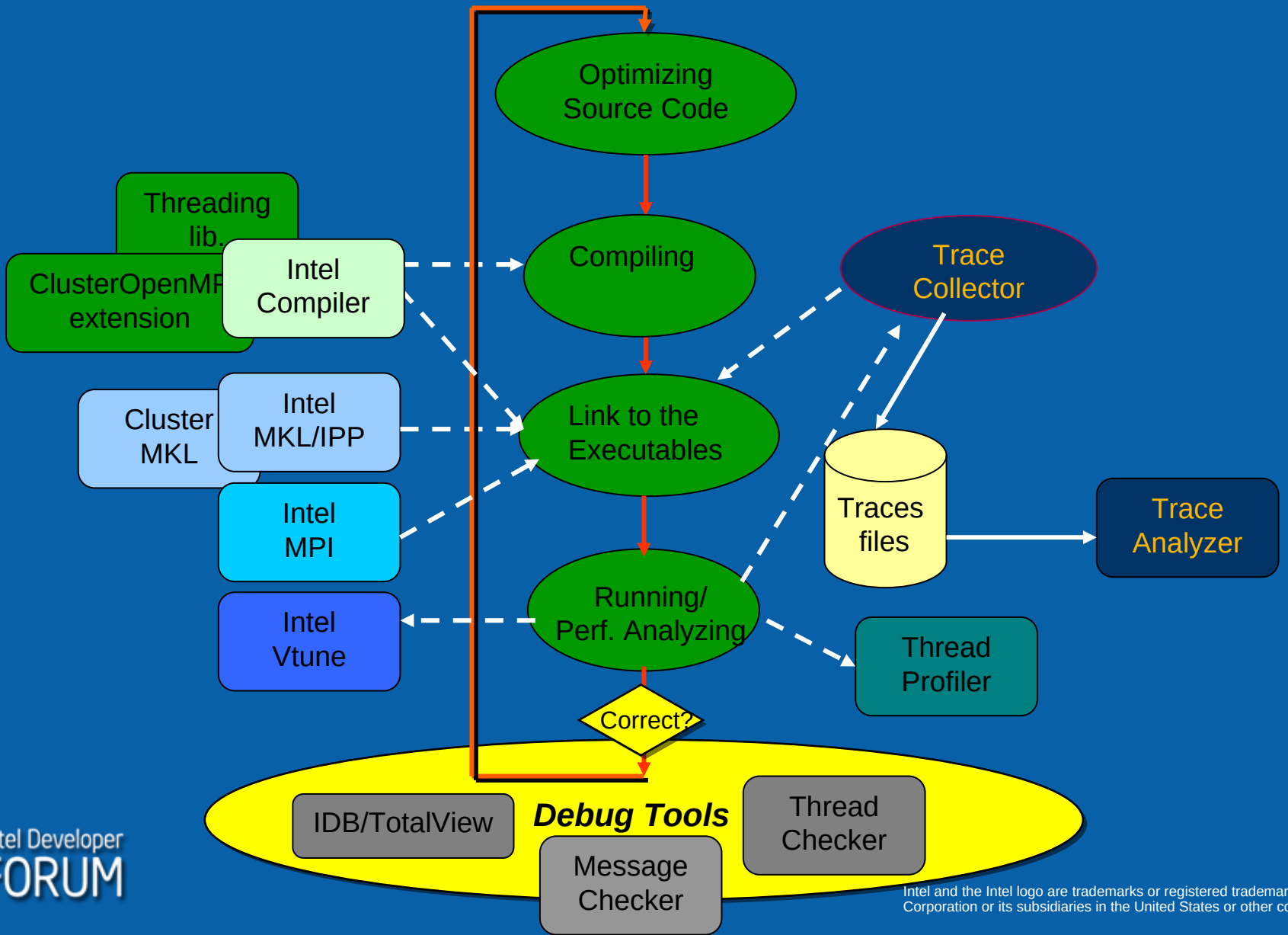
- When integrating Multithreaded code session into the MPI model, we got

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

  The MPI communication time changed to $T'_{mpi,}$ Usually $T'_{mpi} < T_{mpi}$ due to different partition. The parallelized work are roughly scaled out to $N'_{MPI} * N_{thread} * TR\%$ fork-join jobs. Here N'_{MPI} is new MPI processes number. The ΣT_{th} stands for all overhead that threads bring into the system, which may including but not limited to: thread life cycle overhead, FSB latency/bandwidth overhead, share cache coherence overhead, Snoop filter overhead, threads sync. time etc.

- To be simplified, when $T'_{mpi} + \Sigma T_{th} < T_{mpi,}$ We can get overall benefit by introducing Multithread into SMP node. Or we have to fine-tune OpenMP to get more parallelism

Hybrid programming tools: Compiler / ITAC / ITPT

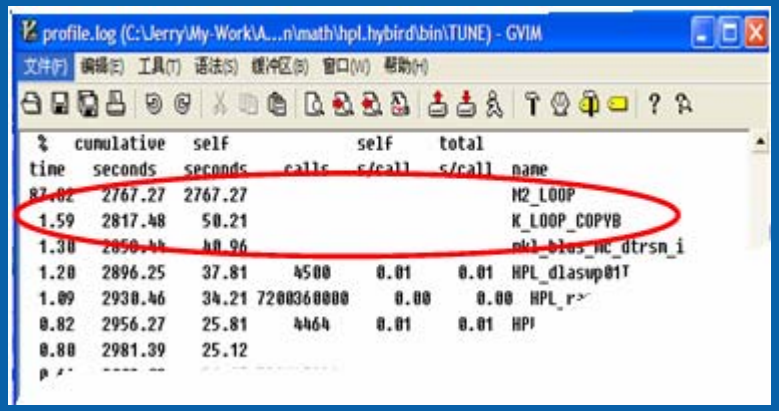


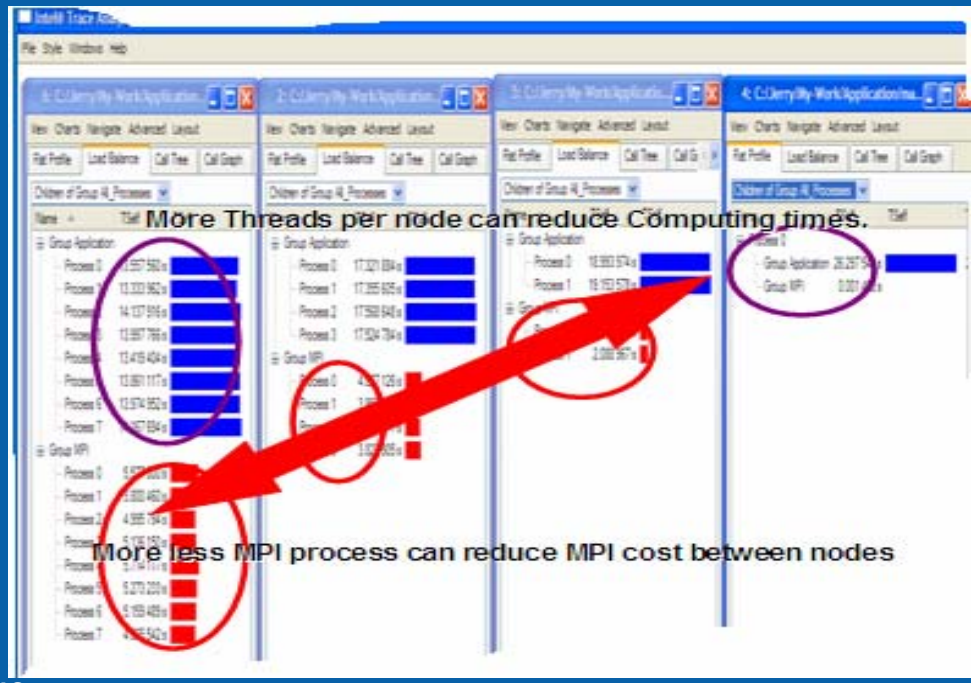
Intel Developer FORUM

Apply Hybrid partition to HPL

- Compare pure MPI, pure OpenMP and MPI+OpenMP version, get slightly gain from hybrid model.
- OpenMP in SMP nodes can really reduce MPI communication cost, but if it's deserved to whole perf.?

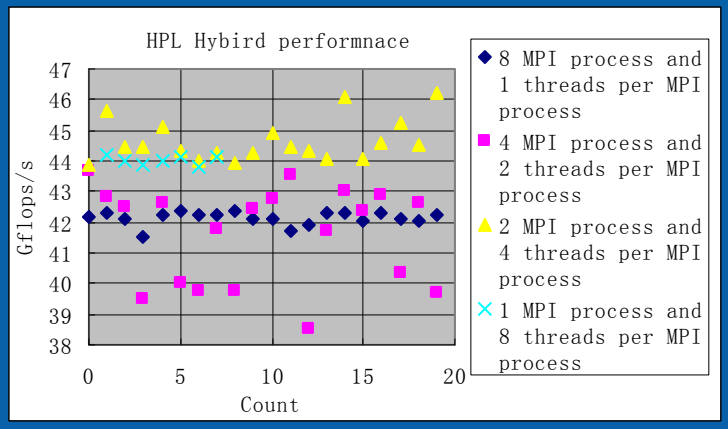
	Core number (MPI process x Threads)	Performnace (Gflops/s)
MPI performance	8 x 1	42.48
OpenMP Performance	1 x 8	44.21
Hybird Performance	2 x 4	46.43

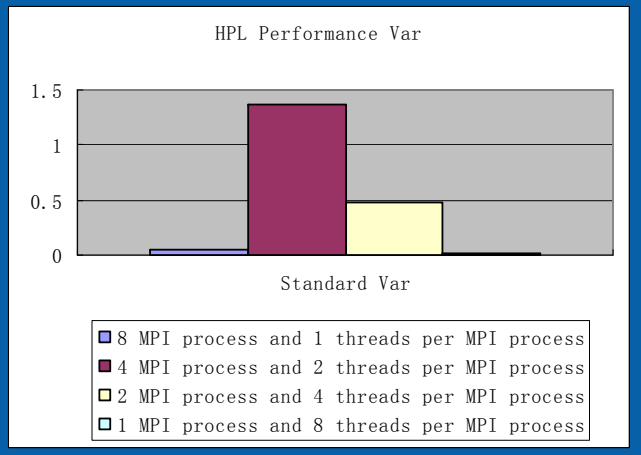




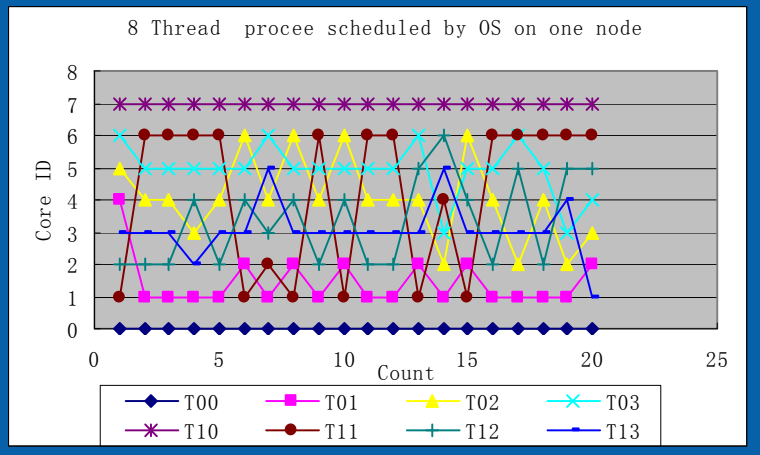
Intel Developer FORUM

Performance statistic dithering due to threads drifting among processors



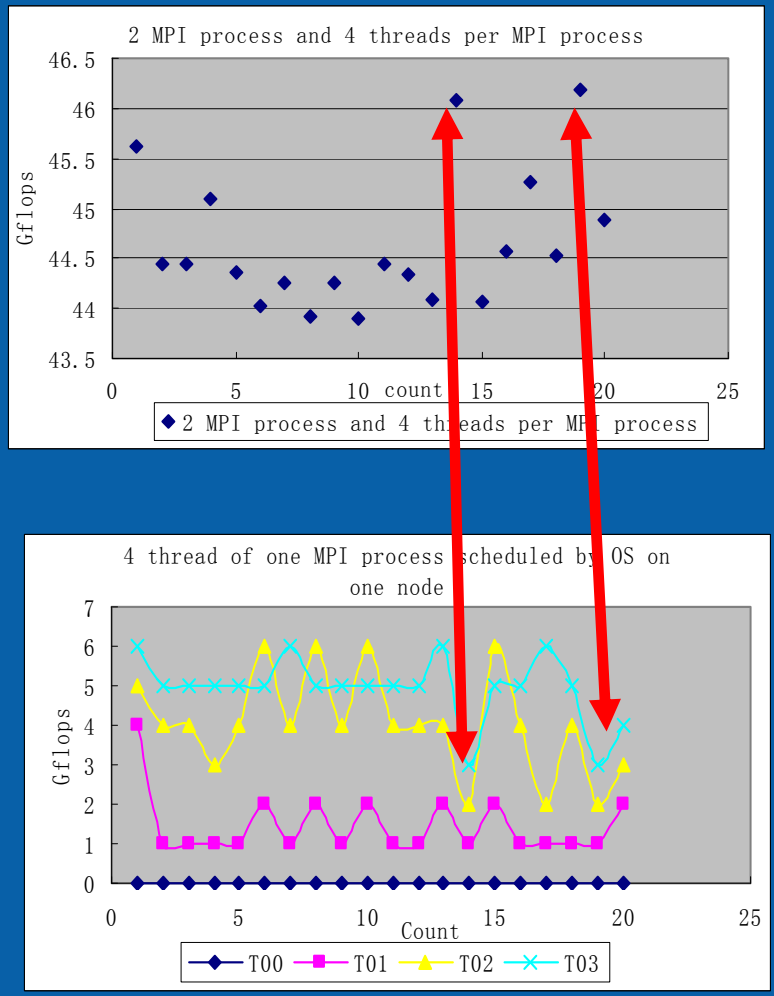


- Performance has statistic dithering in each threading application.
- Unless bundle thread to processor core), dithering is inevitable
- Threads jump between processors in run time

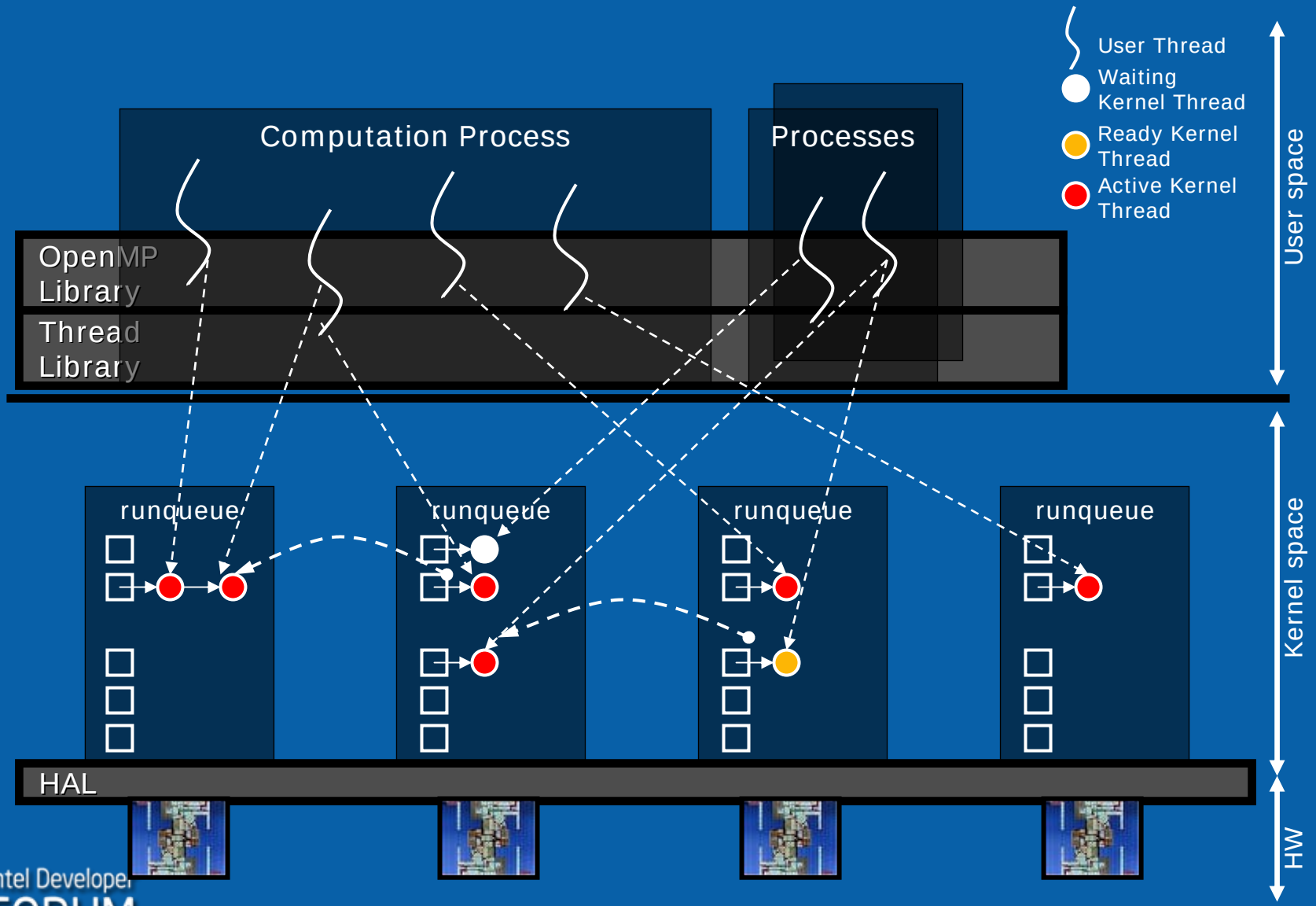


Threads drifting detected by OS thread scheduling

- When bundling MPI process on the Core 0 and Core 7. The other threads will be scheduled by OS.
- From T00 to T03 is created by the No.1 MPI process. T00 is main thread.
- When thread per MPI process was created on one socket, get best performance on Clover-town platform



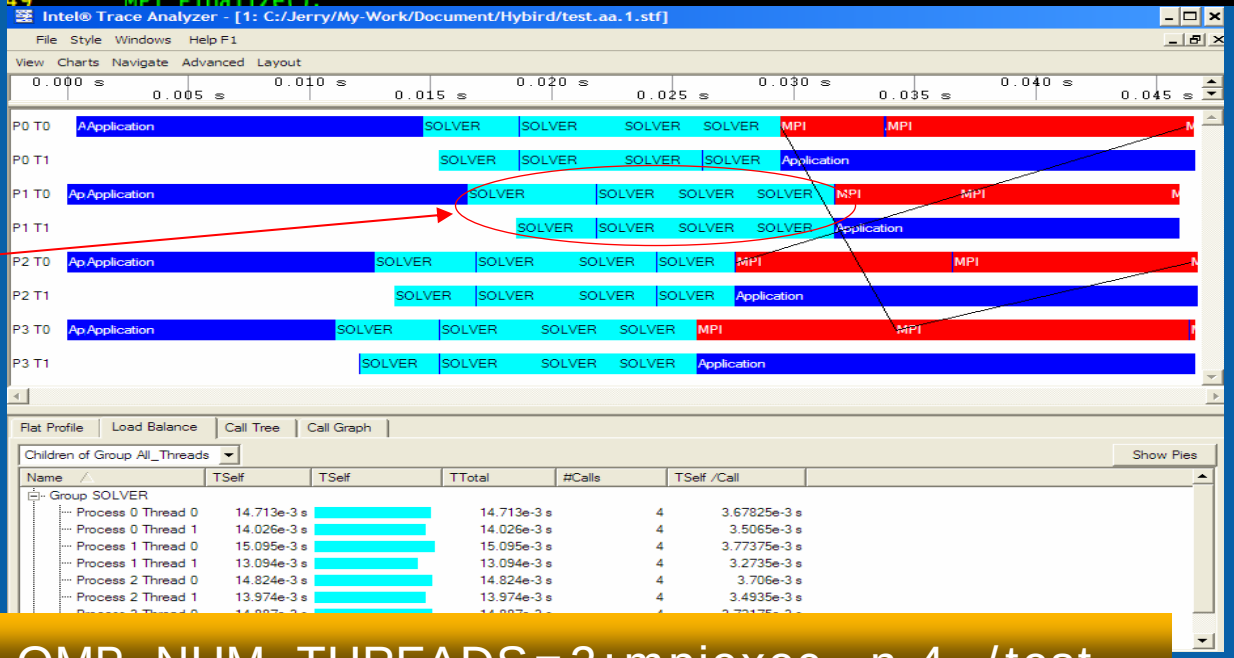
OpenMP threads inside Linux 2.6



ITAC support OpenMP threads tracing by manual adding probe code

- Step 1: Insert function VT_funcdef/VT_enter/VT_leave on source code of the functions which where then executed inside threads
- Step 2: Link Intel Trace Analyzer Tools lib when compiler source code
- Step 3: Run application.
- Step 4: See the trace data with Intel Trace Analyzer Tools after application finish to run.
- Notice the threading parts
- Pay attention to overhead brought by probing code inserted

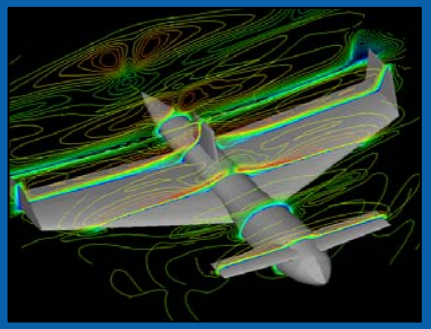
```
20 //        Set VT enter
21          VT_funcdef("SOLVER:execute",VT_NOCLASS,&statehandle);
22          VT_enter(statehandle,VT_NOSCL);
23 +-- 13 行: for (k=0;k<size;k++) {----------------------------------------
36 }
37          if ( myid%2 == 0 ) {
38              MPI_Send(x,1000000,MPI_FLOAT,dest,100,MPI_COMM_WORLD);
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();
```



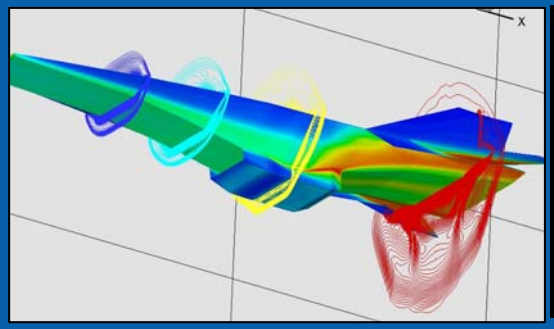
Export OMP_NUM_THREADS=2;mpiexec –n 4 ./test

NAPA Hybrid parallel: Tune both MPI and OpenMP

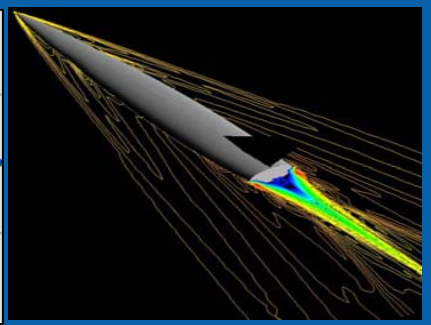
- Napa* is a NUAA self-developed CFD application, The code has been fully MPI-tuned by Intel
- MPI version has lots of point-to-point communication to update the data of ghost grids every iteration, and has no much collective communication
- MPI version Only get no more than 60 speedup on Ethernet, low scalability. This is a good candidate to apply hybrid parallel model.



MAV



X-43



NASA Lift body

* NAPA - NuAa Program for Aerodynamics, developed by NUAA and optimized by Intel/NUAA

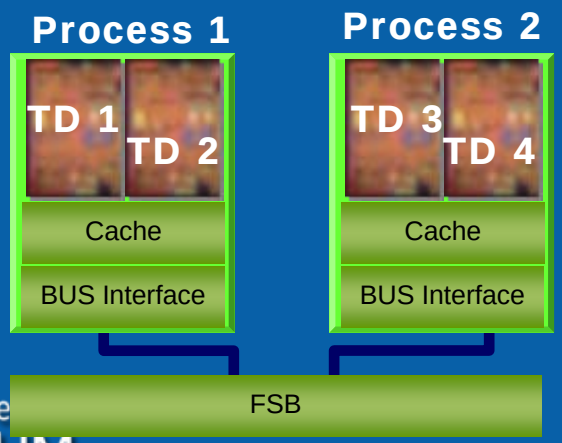
MPI tuning: Consider both Computation and Communication

- Overlapping Computation and Communication?
  - For general MPI implementation (Mpich1.2, Intel mpi 1.0) in the Share memory, Ethernet, Infiniband, Numa-link, the answer is No.
  - Some specific Message Passing Structure, the answer is Yes.
    - For example: QCDOC Supercomputer and its LQCD Message Passing API (QMP)
  - NAPA parallelization complete for cluster, validate with benchmark results.

Simple OpenMP vs. MPI

- benchmark result on Xeon 5100 1 nodes (with 4 cores)
  - mpich 1.2.7p1 compiled with Intel compiler
  - Intel Fortran compiler 9.1.033

cores / speedup	1 cores	2 cores	4 cores
Serial	1	-	-
OpenMP	-	1.77	2.8
MPI	-	1.76	3.14 (4 MPI process)
Hybrid	-	-	3.10 (2 threads/process)



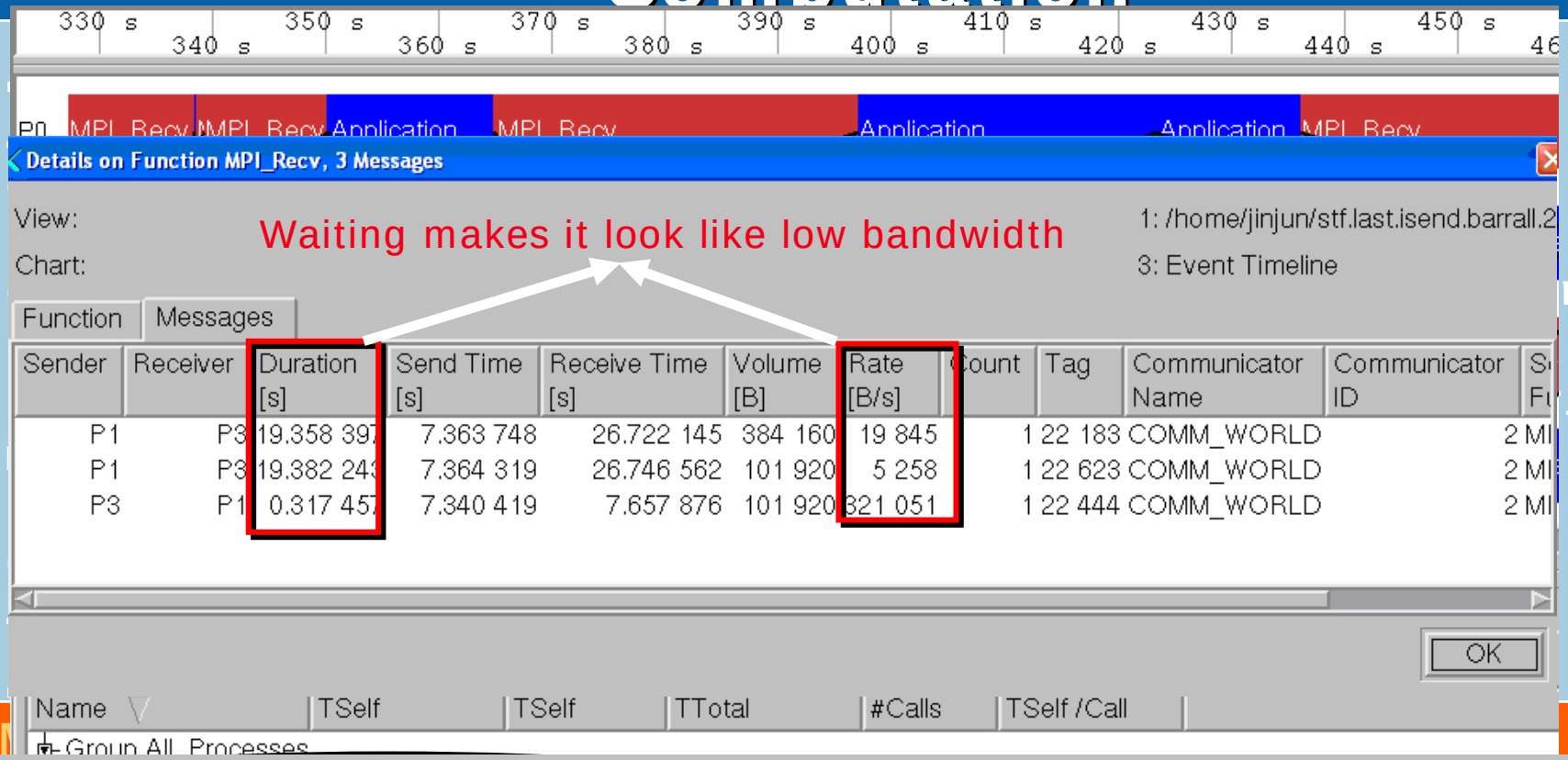
Hybrid Model

OpenMP scale not so good from 2cores to 4 cores, the overhead $T_{th} > T_{mpi}$

MPI has better performance on 4 cores

NAPA: MPI Communication and Computation

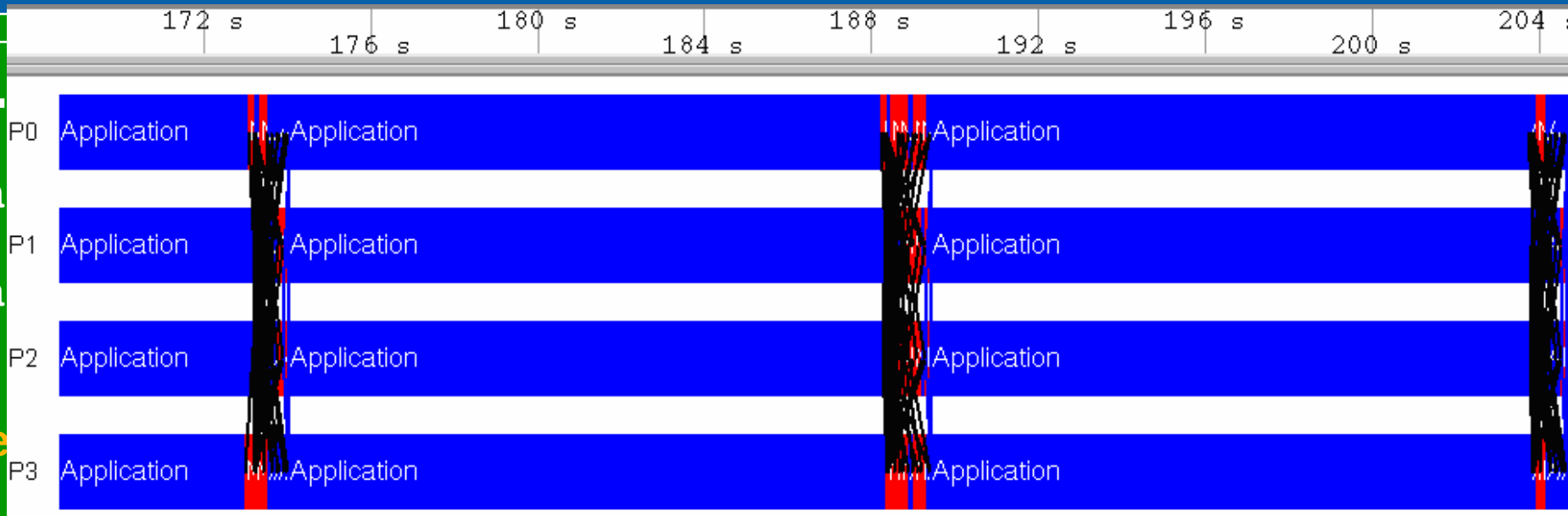
Grid:97*97*97*24



Process 0 | MPI_Isend | MPI_Recv (waiting for 1's send) | Computation | MPI_Wait

Process 1 | MPI_Isend | MPI_Recv | Computation | MPI_Wait | ..Next iter..

MPI Communication and Computation (cont.)



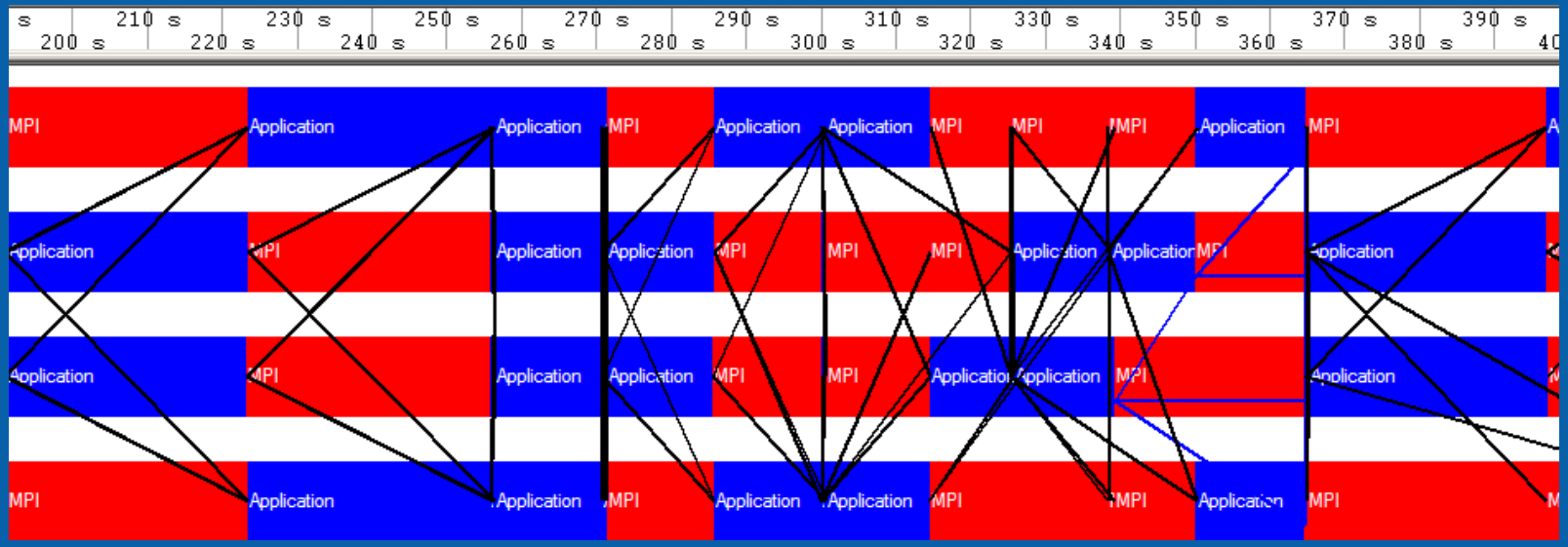
Flat Profile | Load Balance | Call Tree | Call Graph

Group All_Processes

Name	TSelf	TSelf	TTotal	#Calls	TSelf /Call
Group All_Processes					
Group Application	96.05 %		100.00 %	0	
Group MPI	3.95 %		3.95 %	312	0.01 %

Performance has been improved from 6m52s to 4m03s, 1.45X speedup

NAPA: MPI Parallel optimization apply to real workload



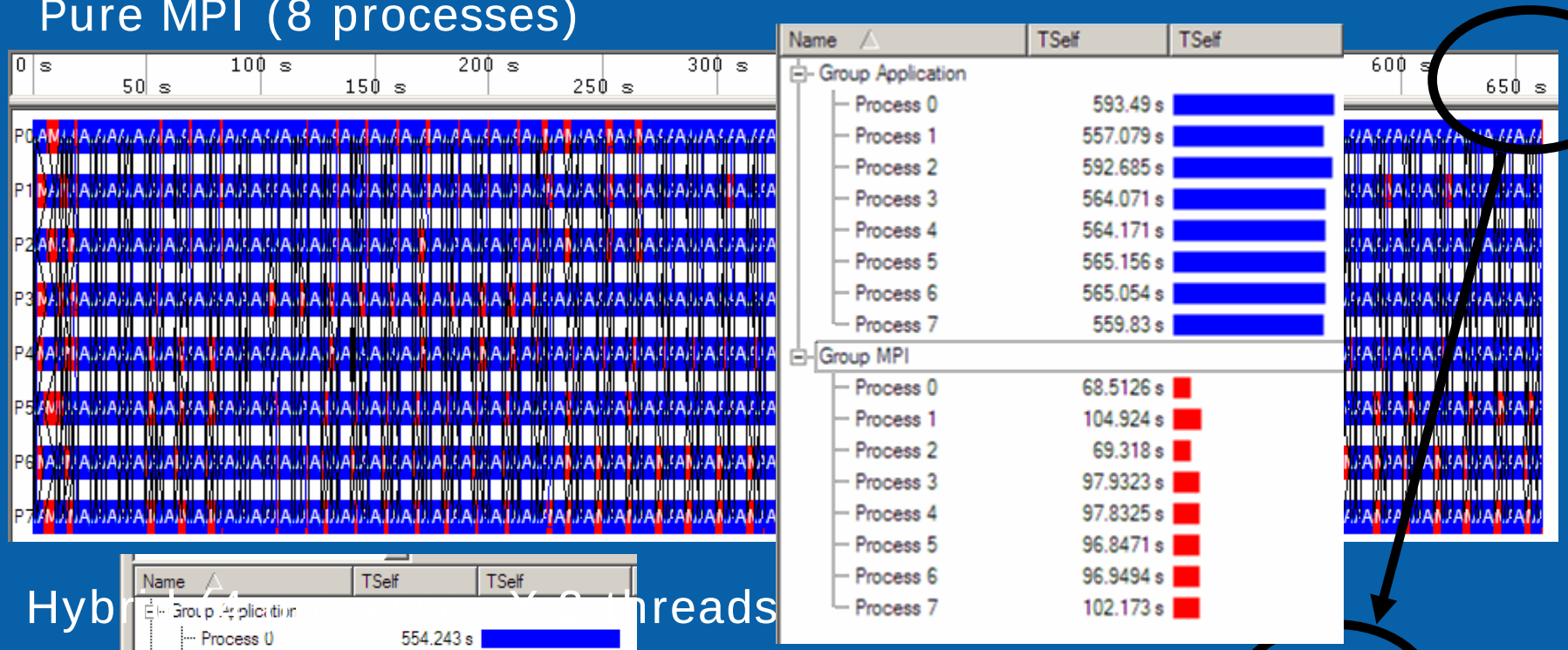
Workload: Ultra-sonic in-let
4 processes

Performance boost 41.1%

Intel Developer FORUM

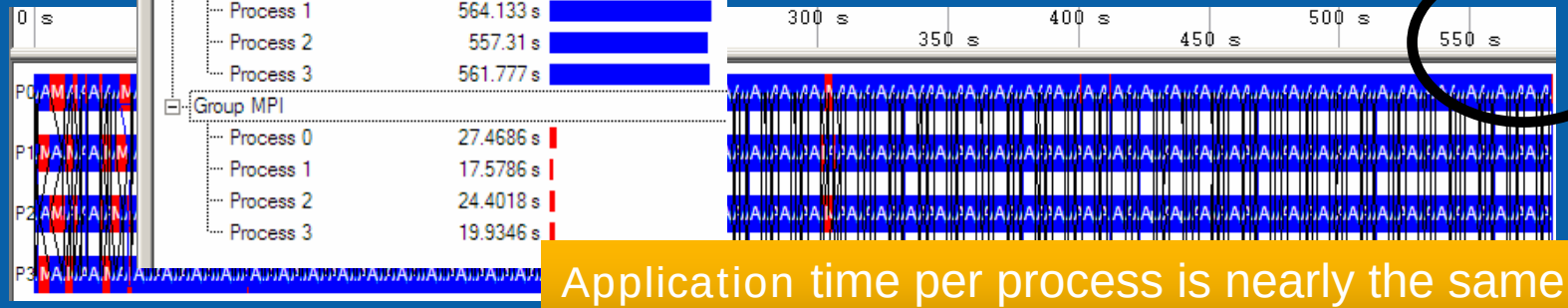
Threads reduce the computation

Pure MPI (8 processes)



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	

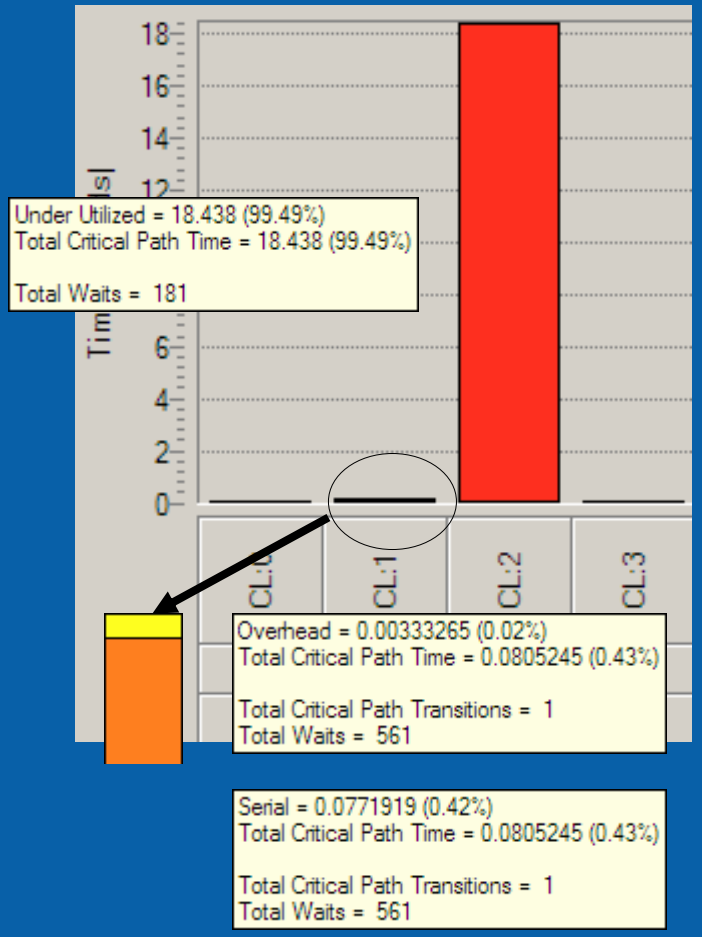
Hybrid (4 processes X 2 threads)



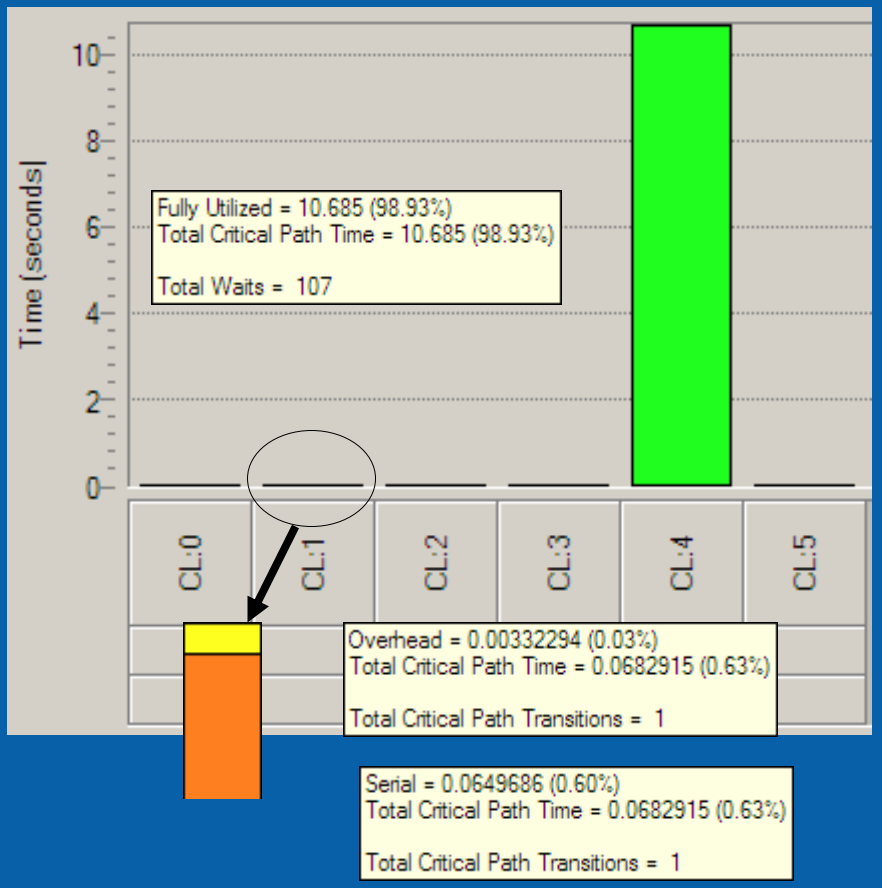
Name	TSelf	TSelf
Group Application		
Process 0	554.243 s	
Process 1	564.133 s	
Process 2	557.31 s	
Process 3	561.777 s	
Group MPI		
Process 0	27.4686 s	
Process 1	17.5786 s	
Process 2	24.4018 s	
Process 3	19.9346 s	

Application time per process is nearly the same in Hybrid model, the communication is greatly reduced from 13.9% (PureMPI) to 3.84%(Hybrid)

Profile OpenMP thread by Intel threading profiler



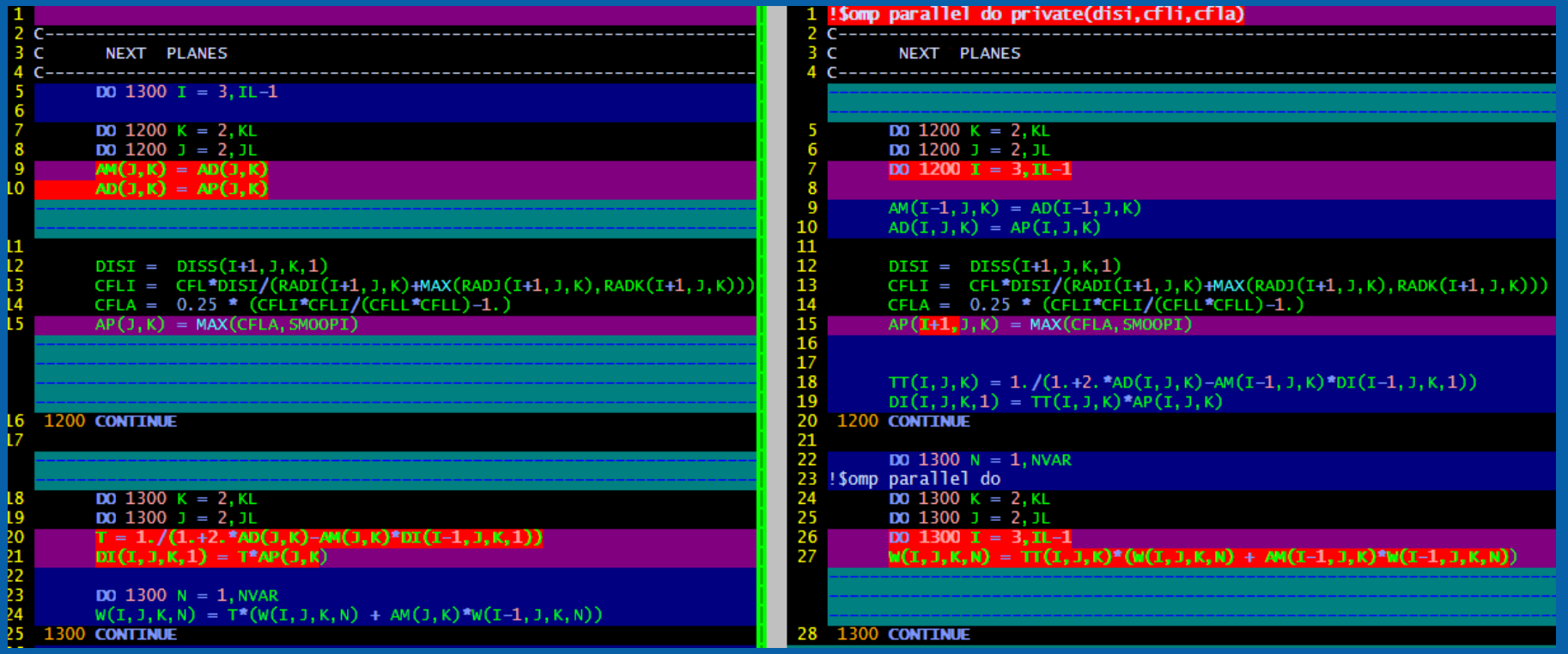
2 threads



4 threads

Intel Developer FORUM

After MPI tuning, avoid naïve OpenMP – fine tune it.



```
      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)
      NEXT  PLANES

      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
```

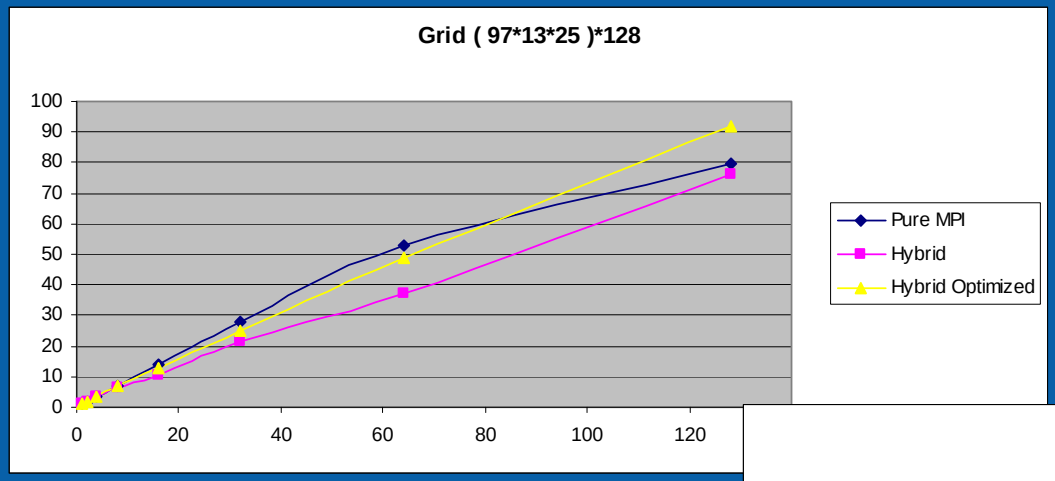
Fine tune the OpenMP part, make its grain more coarse and decouple the data dependence in loop

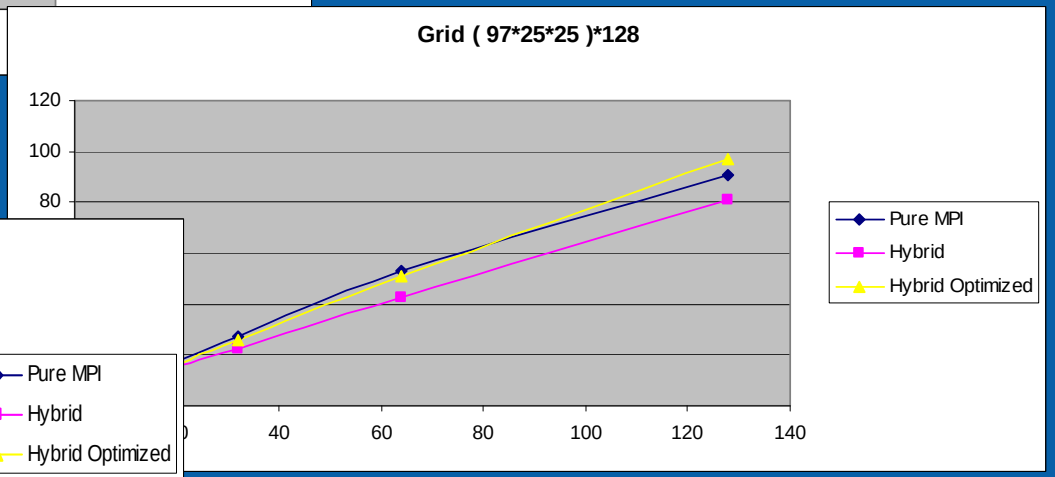
Intel Developer FORUM

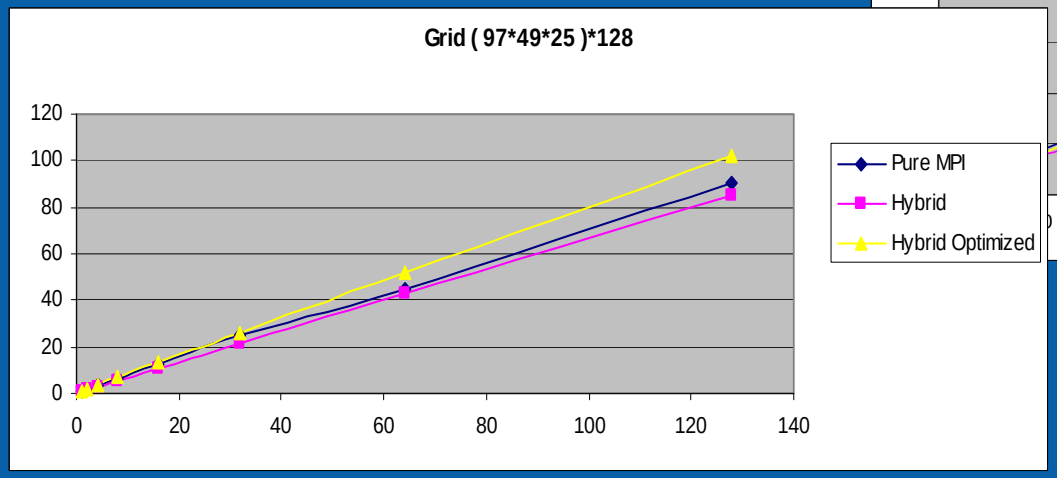
Benchmark Result: Hybrid boost speedup based on MPI

Hybrid: 2 threads get 1.6~ speedup

Hybrid opt: 2 threads get 1.77 speedup



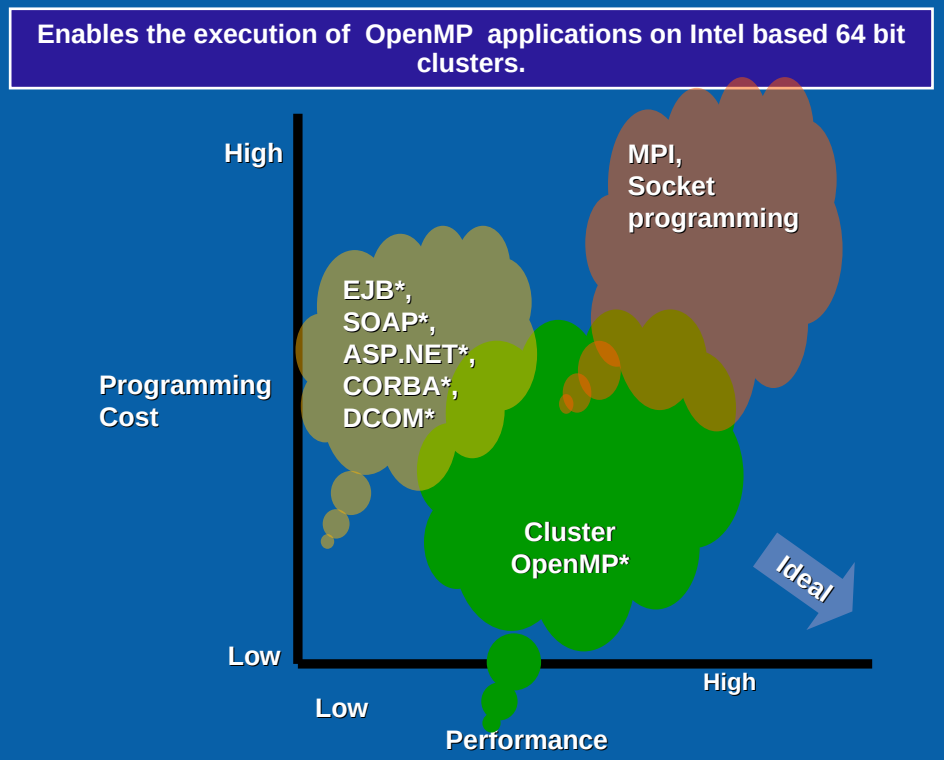




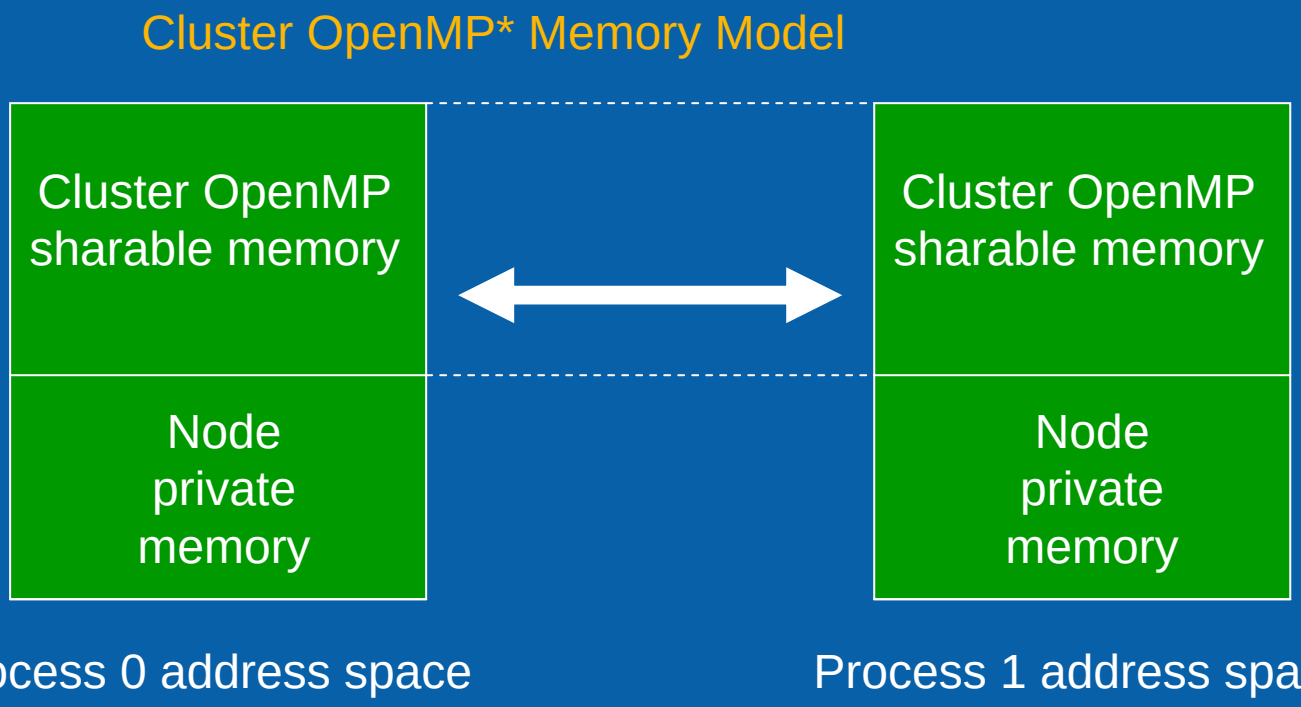
- For hybrid, every process contain 2 threads
- MPI benchmark implemented Share-Memory on same Node

Intel Cluster OpenMP as "Hybrid" model: A tool to avoid MPI complexity

- Intel Cluster OpenMP is an OpenMP extension to apply OpenMP threads across Cluster nodes.
- Cluster OpenMP* is an easier way to program clusters. Certain codes can achieve good performance and scaling with Cluster OpenMP. The process of porting a code to Cluster OpenMP is systematic and much less work than the porting process to a message-passing form. Performance tools help tune the program for best performance on a cluster.

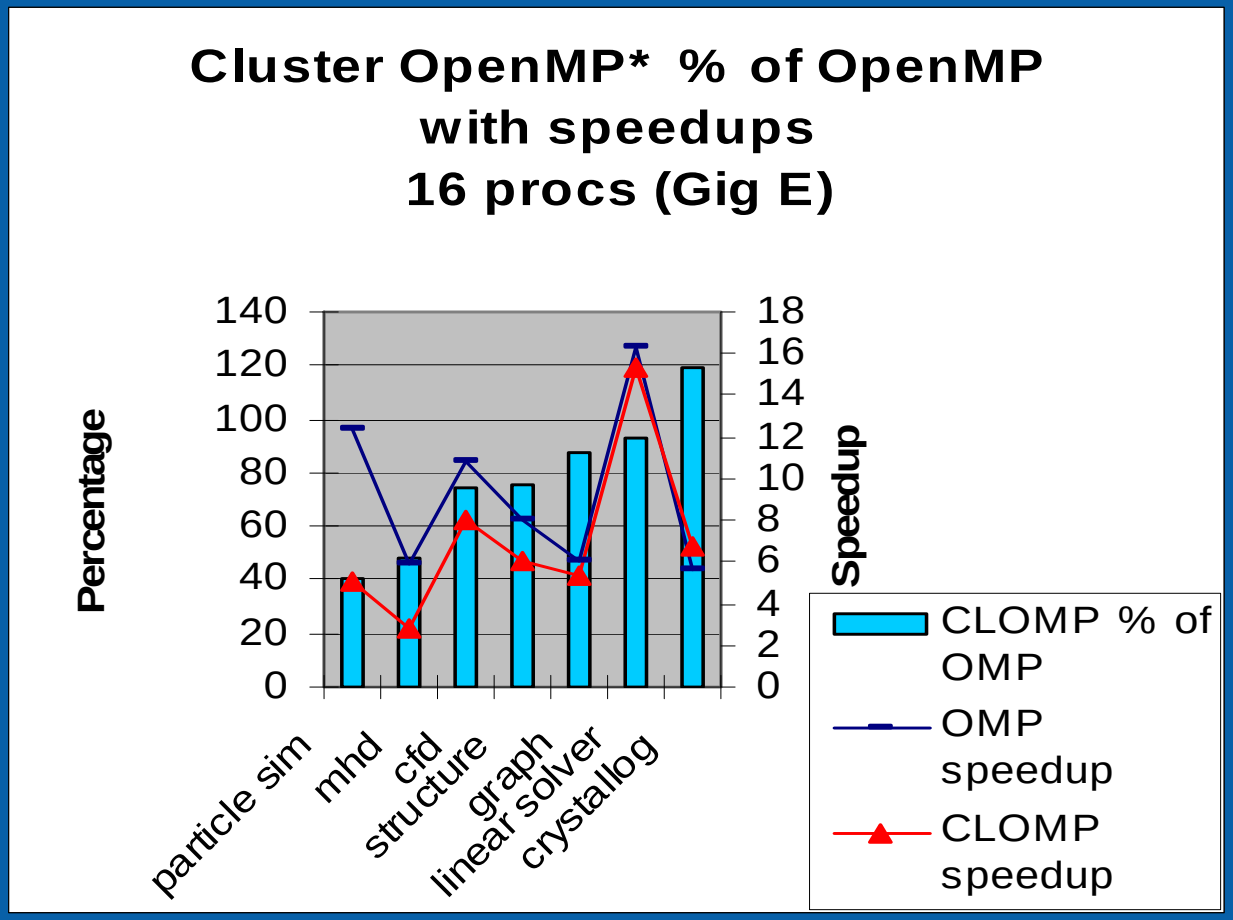


How does Cluster OpenMP work?



Cluster OpenMP is implemented by compiler extensions and a run-time library that supports running an OpenMP program on a cluster. The ClusterOpenMP memory model create a virtual share-memory region across the distributed nodes, which make it easy for users to avoid MPI programming while take advantage of distributed memory.

Scalability of Cluster OpenMP in Hybrid Architecture

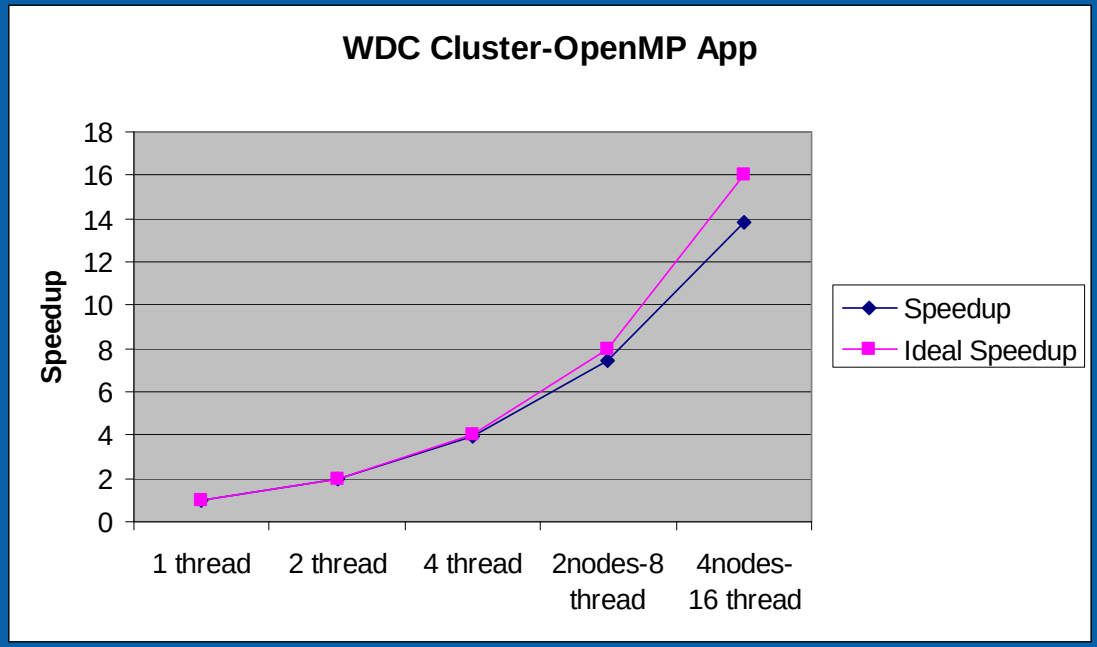


* Other names and brands may be claimed as the property of others.

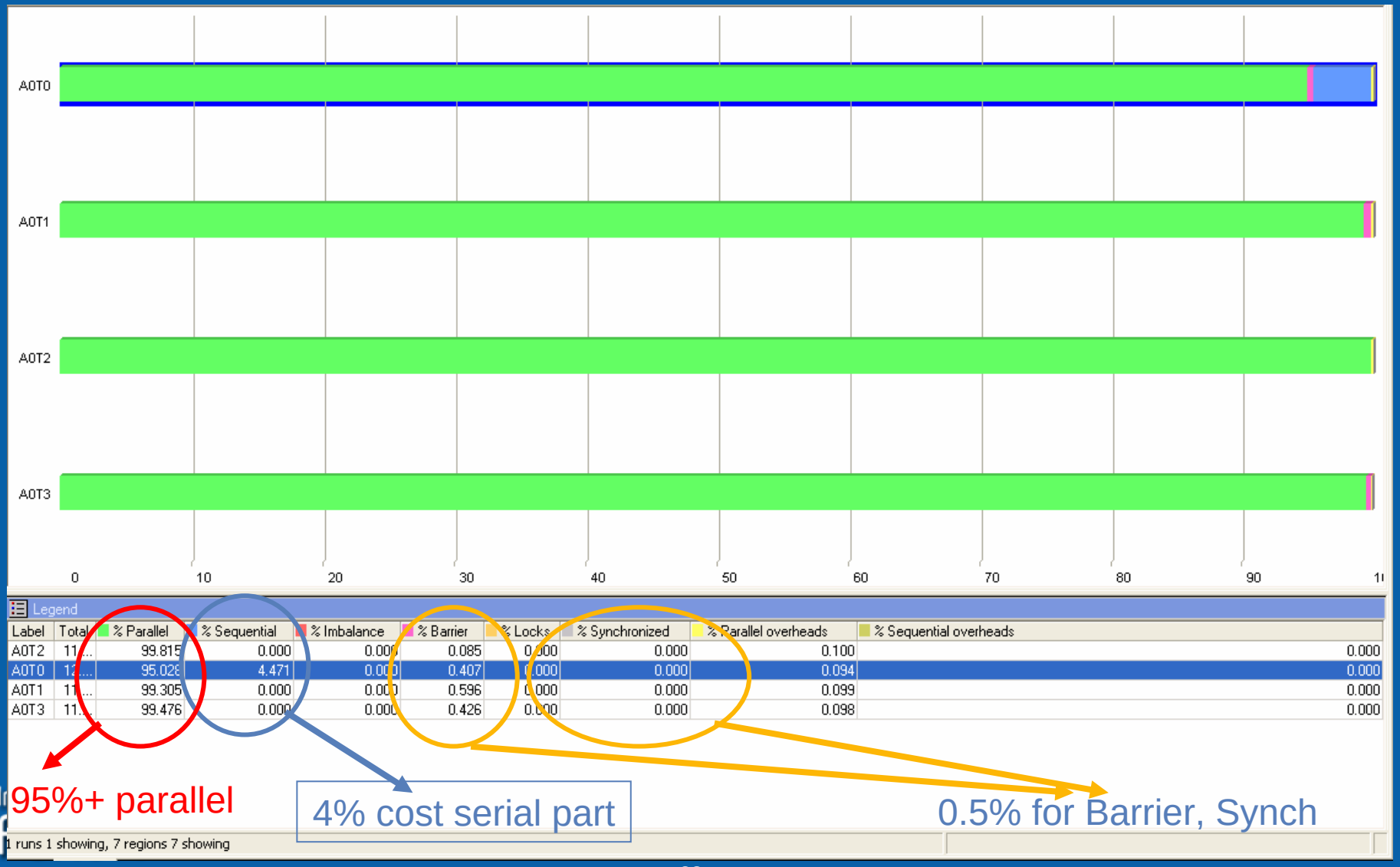
Case Study: GalRen in 2-way Xeon 5100 Hybrid Cluster

- Main function: rendering a movie of a flight through space.
- It uses a database of stars (brightness, color, position, etc) and does ray-tracing through them.
- Each thread takes a set of scan-lines of the frame buffer and computes the pixels.

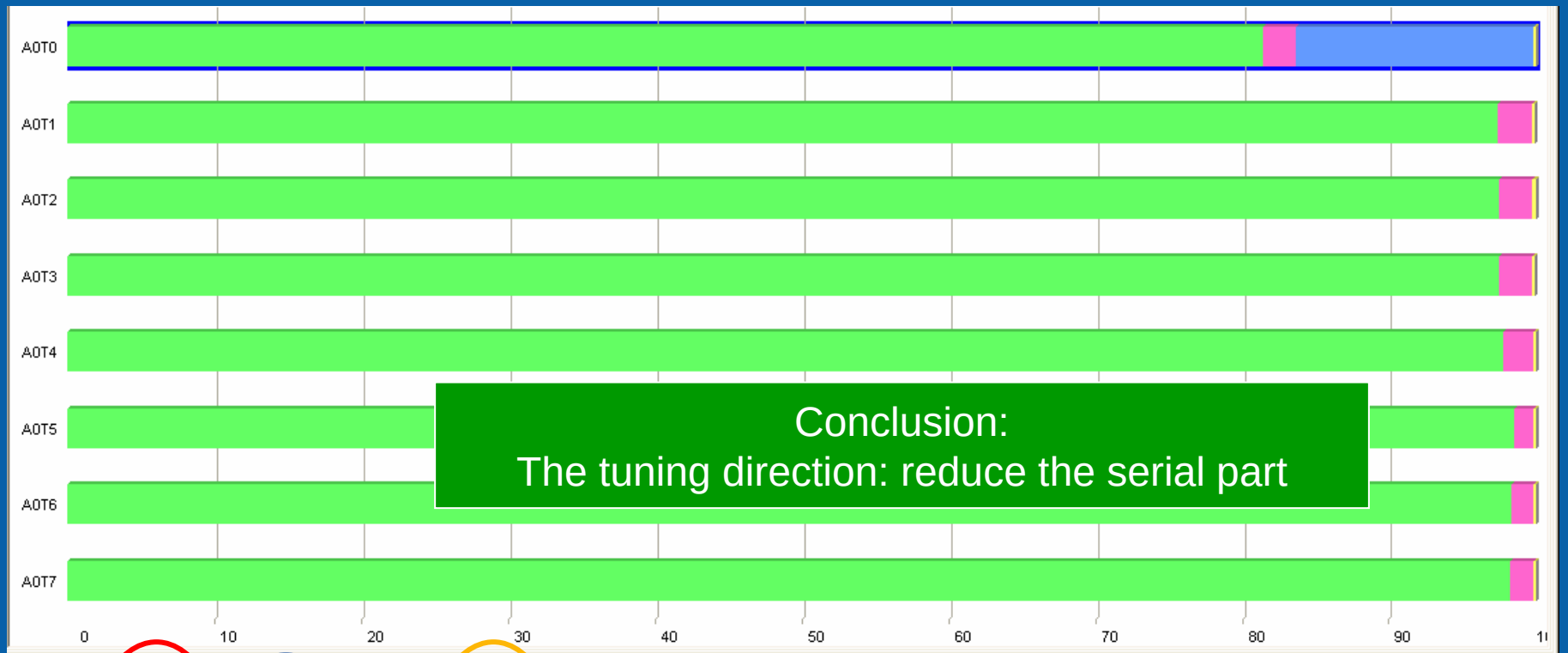
14X Speedup for 16 Threads



Case Study: GalRen:4threads in one node



8threads in 2 nodes



Legend

Label	Tota	% Parallel	% Sequential	% Imbalance	% Barrier	% Locks	% Synchronized	% Parallel overheads	% Sequential overheads
A0T0	8.420	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	7.050	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	98.268	0.000	0.000	1.560	0.000	0.000	0.175	0.000

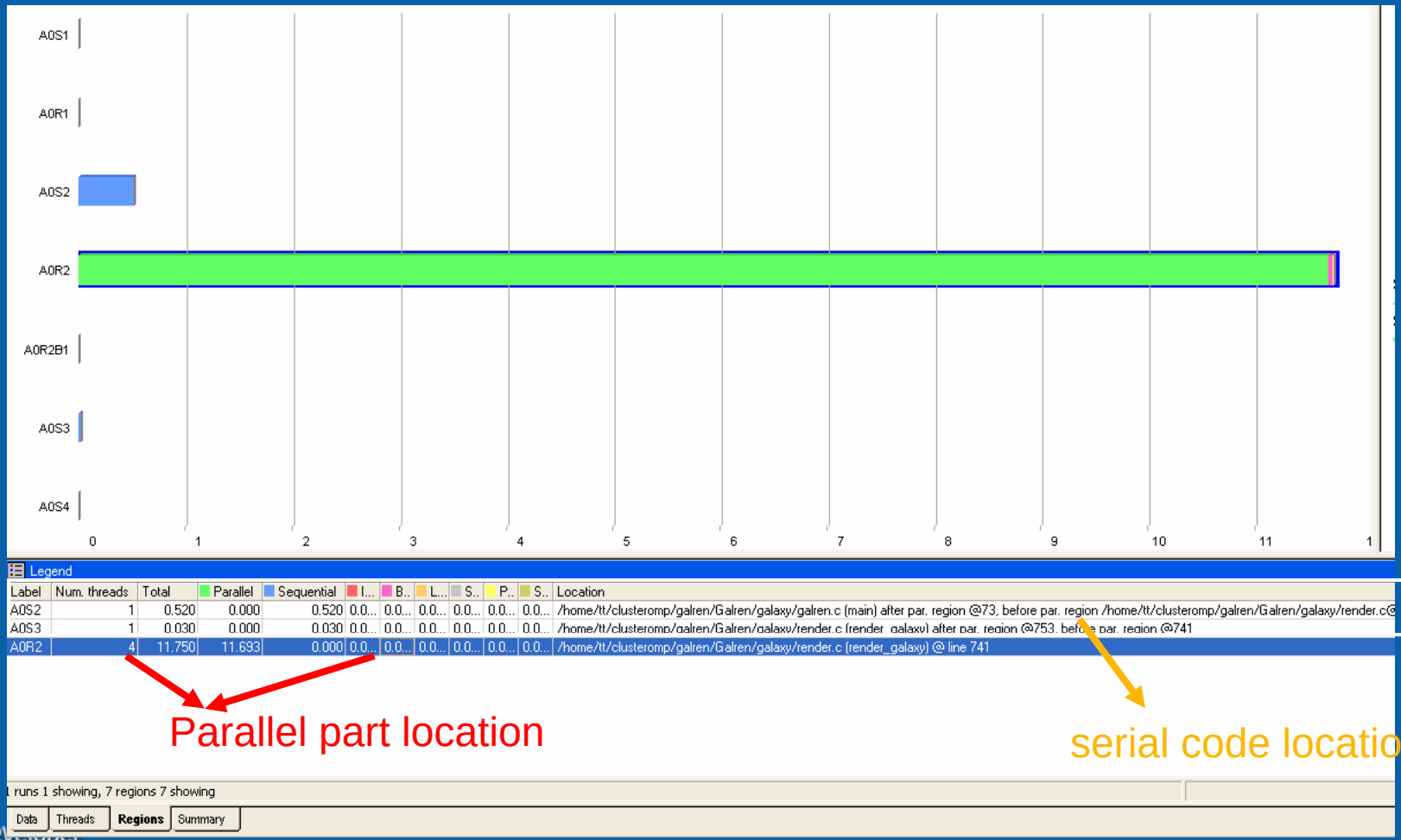
81%+ parallel

16% cost for serial part

2.5% for Barrier, Synch Part

How to find out the source code: Location vs Threads

4threads in one node



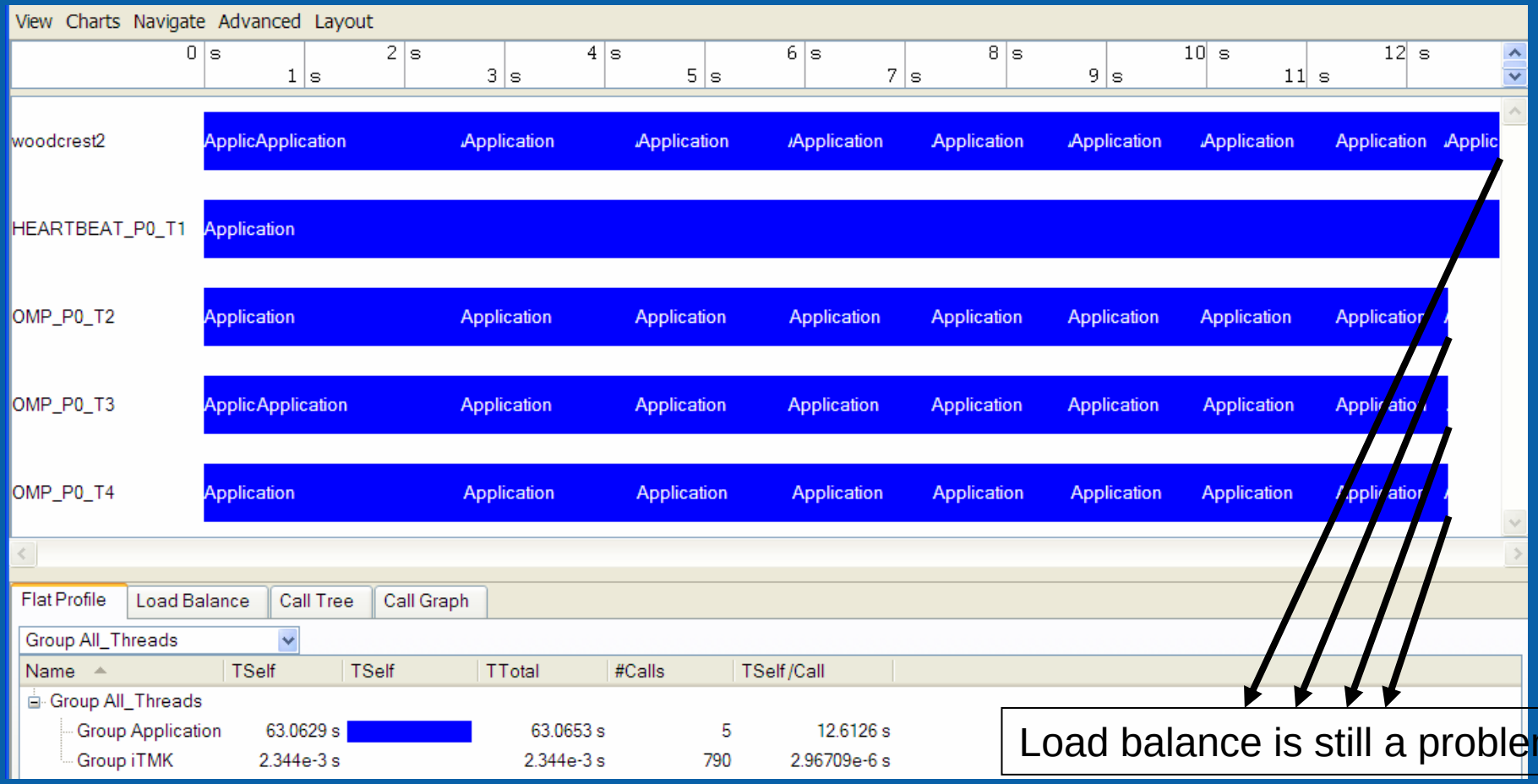
```
#pragma omp parallel private(i, j, R, G, B, A)
   {
#if defined(OMP)
      int thread_num = omp_get_thread_num();
      int my_slice = xsize / omp_get_num_threads();
      int my_offset = (thread_num * my_slice);
      int my_bytes = ( sizeof( uByte8 ) * my_slice );
      uByte8 *my_pixels;

      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
}
```

One Part of Cluster OpenMP Pseudo Code

Very Little Change needed

ITC/ITA(4 threads*1 nodes)

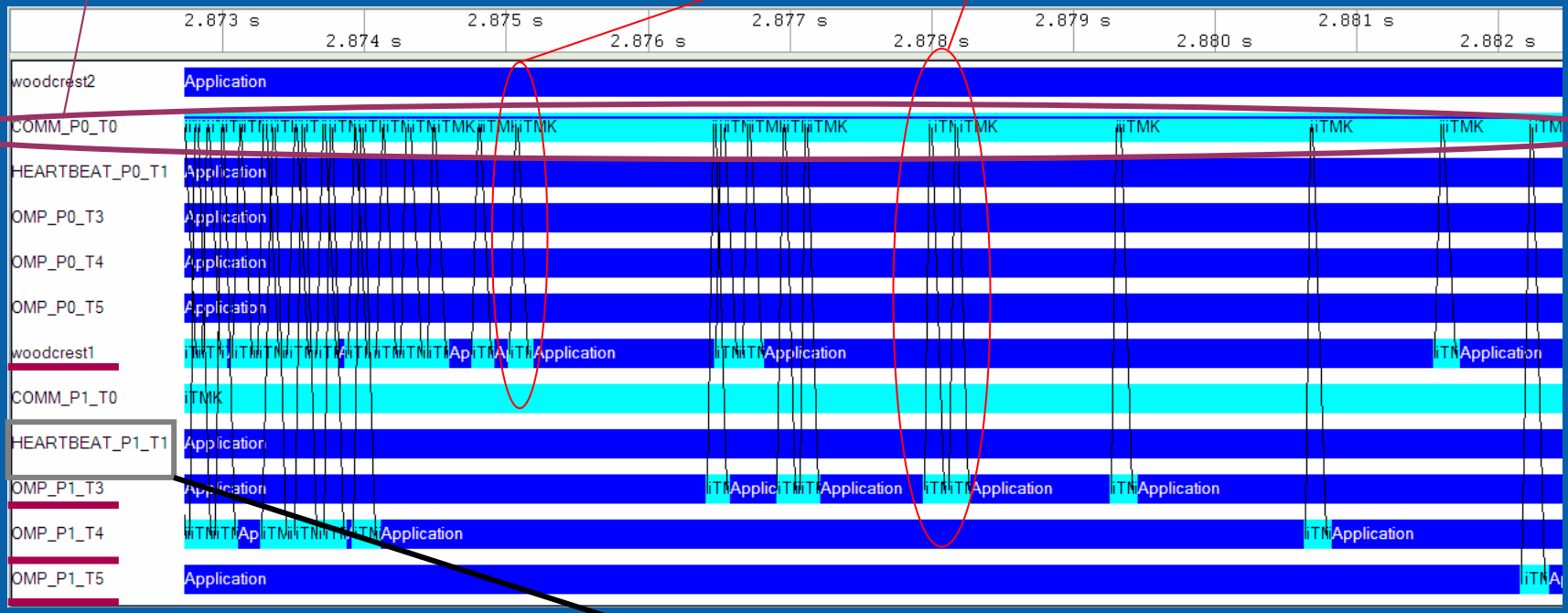


Almost Non-Communication part in single node

ITC/ITA(4 threads*2 nodes)

COMM_P*_T0 Thread deals communication with threads in other nodes

Message only between nodes



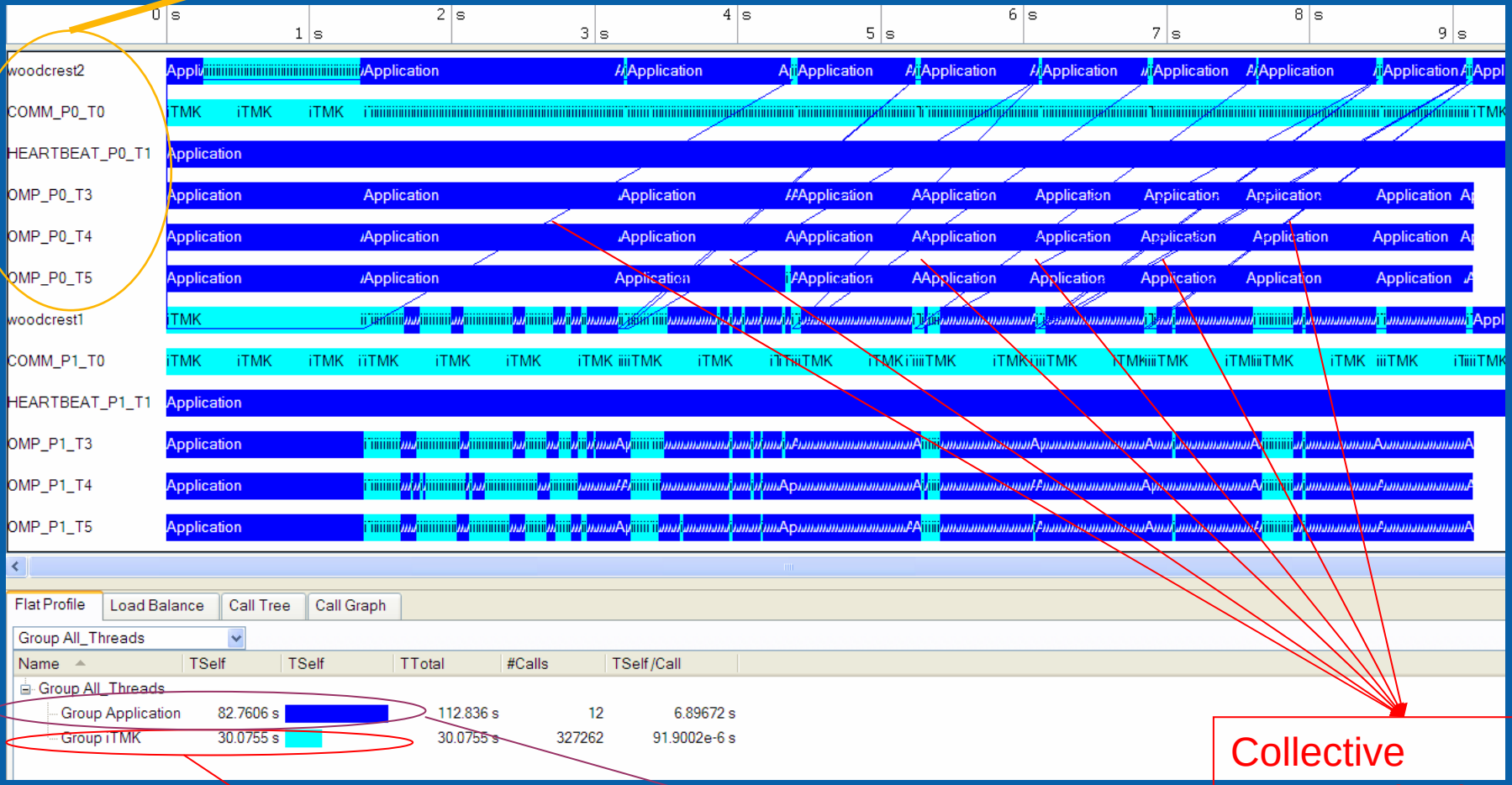
Main thread and the other ***3 slave threads*** in one node ***for computation***

Heart beat Thread each node

Intel Developer FORUM

ITC/ITA(4 threads*2 nodes)

More threads get better overlap for communication and computation



Collective Communication

Communication part

Computation part

Summary/Call to Action

- Consider both threading and MPI parallelism to maximize scalability in current Multicore SMP Cluster.
- Tradeoff between MPI and OpenMP, use Intel ITAC and threading tools to Analyze the threads' behavior and overhead.
- Considering parallel granularity, apply MPI tuning (large granularity) before introducing fine-tuned OpenMP threading.
- Use Cluster OpenMP can fast deploy/debug Multithreading in SMP cluster, get reasonable speedup to certain type of applications without MPI/PVM programming.

Please fill out the Session Evaluation Form

Thank You for your input, we use it to improve future Intel Developer Forum events

Save your date for IDFs this Fall:
San Francisco, USA 2007 September 18 -20
Taipei, Taiwan, 2007 October 15 -16

Intel Developer FORUM

Thanks!

- Thanks for ICSC CRT/HPC enabling team, especially for Wang Zhe, Qiao Nan, Jin Jun and Xu Jin.
- Also thanks for Hoeflinger, Jay P, Meadows, Lawrence F and Ohlay, Patrick for their help in Cluster OpenMP and ITAC threading tracing problem

Backup

Advantages of Cluster OpenMP

- To the user
  - Expand problem size
  - Reduce the cost of computing.
  - Eliminate capacity bottlenecks
- To the developer
  - Reduce development time
  - Improve maintainability
  - Could take advantage of existing OpenMP code
  - The implementation is easy going, no need to master MPI or PVM knowledge.
  - Code is easy to read, keeping consistent with the serial code more than MPI/PVM code.
  - Debug is simple as it's small step based

Solution:

Method #1:

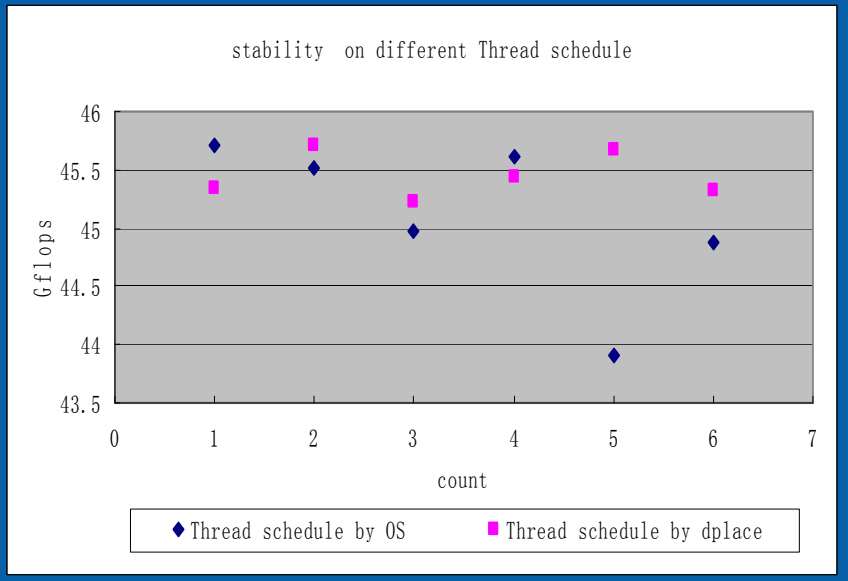
Some OS can supply tools to permit to schedule Thread|Process on different core|cpu. Such as SGI or BULL's NUMA tools.

Method #2:

Insert OpenMP function sched_setffinity on OpenMP region. Such as :

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

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



Benchmark on SGI Altix. Use dplace to schedule threads