ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

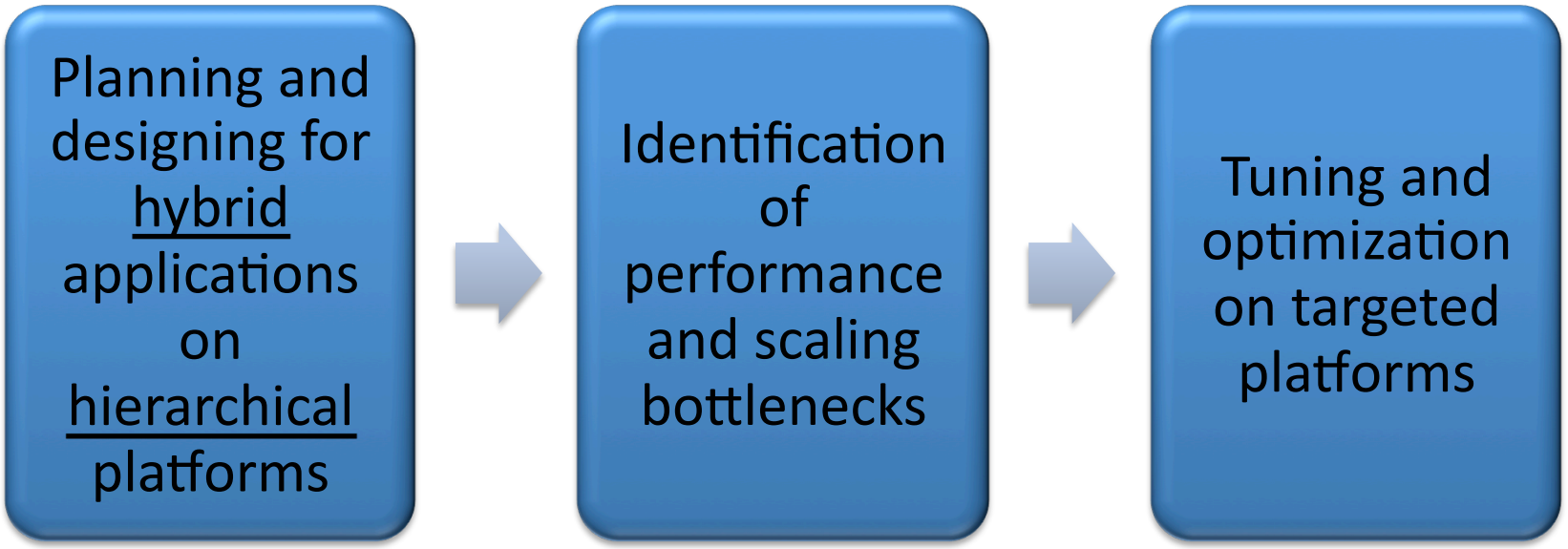
CSCS
Swiss National Supercomputing Centre

DAY 3: Performance Considerations, Best Practices and Misconceptions …

Course on Multi-threaded Programming, Tuning and Optimization on Multi-core MPP Platforms

Feb. 15-17, 2011

Outline


Planning and designing for hybrid applications on hierarchical platforms
Identification of performance and scaling bottlenecks
Tuning and optimization on targeted platforms

PART I—HYBRID PROGRAMMING FOR HIERARCHICAL SYSTEMS

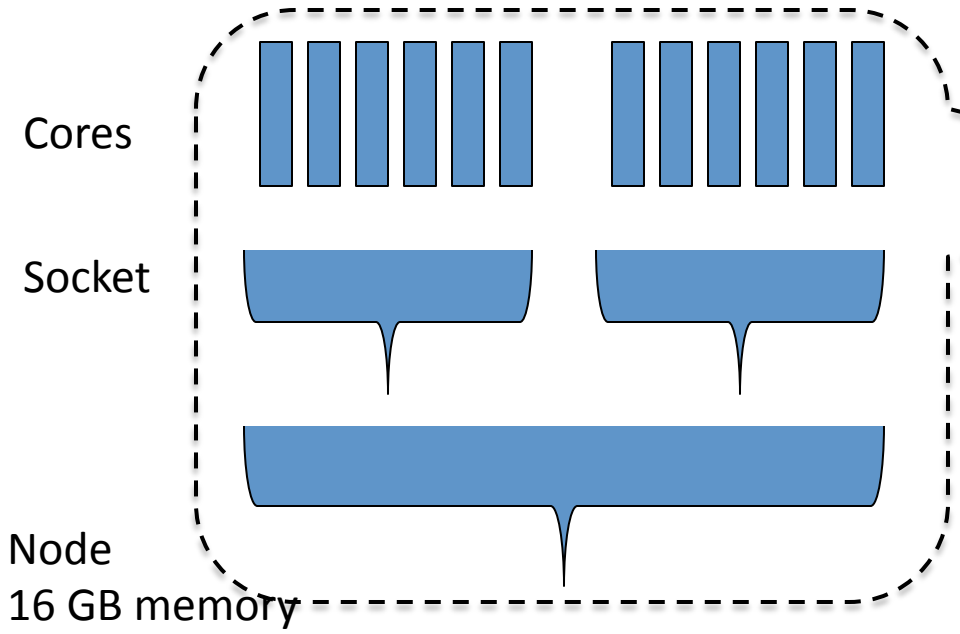
Hybrid MPI and OpenMP Programming

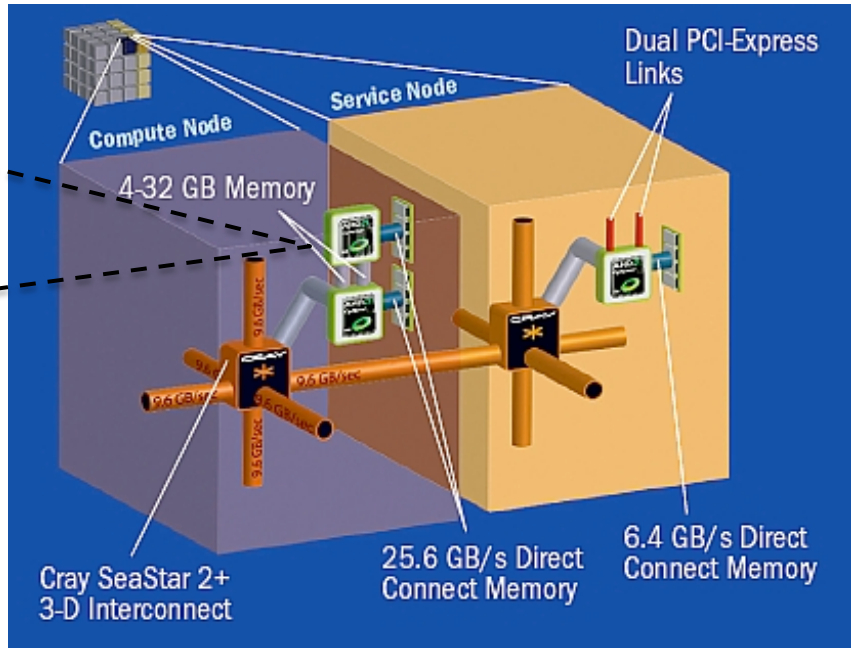
- On large-scale MPP systems with ever decreasing memory/core and rapidly increasing resource contention
  - MPI between nodes—OpenMP on nodes
  - MPI between sockets—OpenMP on sockets
  - MPI on collection of cores—OpenMP on thread groups
- Fortunately on MPP systems, there are largely fixed bindings and affinities
- Unfortunately bindings are currently static—more later…

Hierarchical MPP Systems (Rosa)

Cache levels (L1 & L2 not shared, L3 socket level)

SSE Units (4 FLOP / clock cycle)

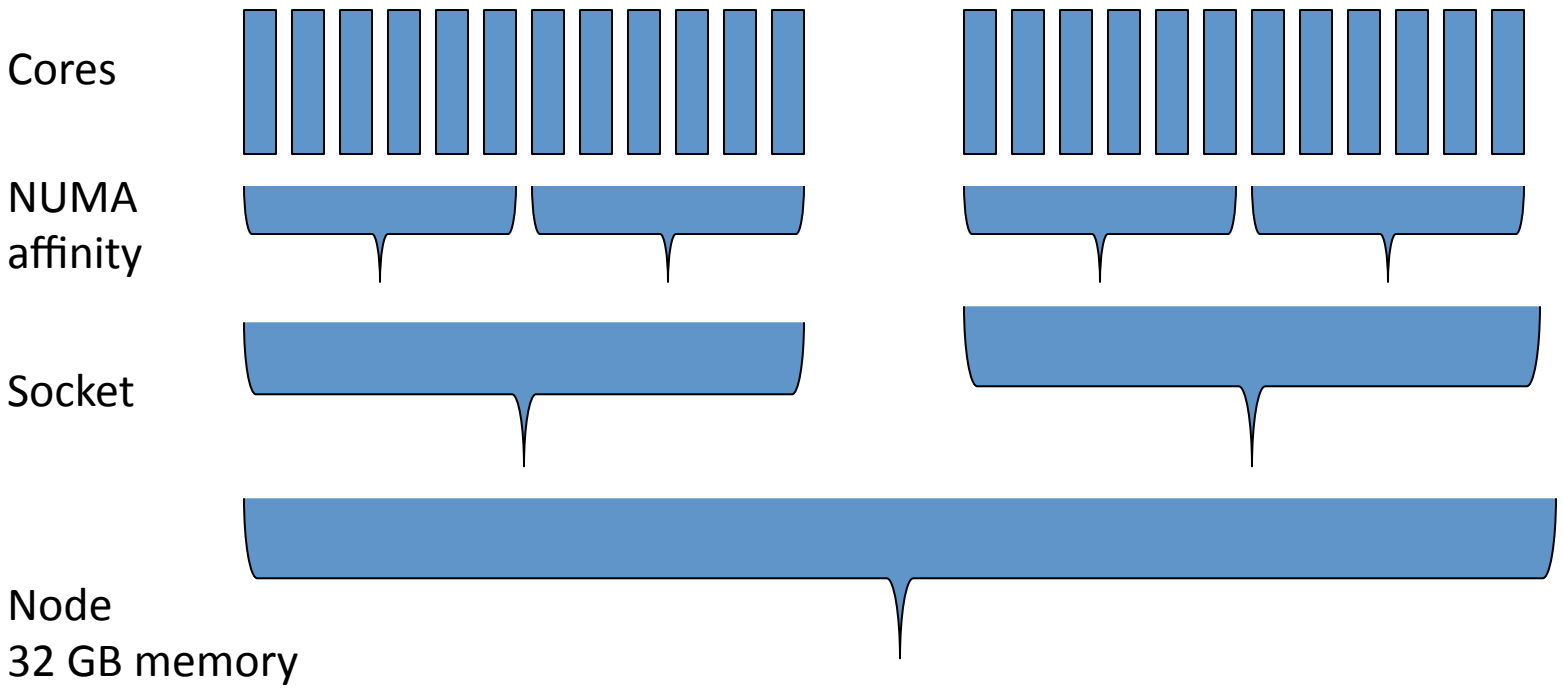




Hierarchical MPP Systems (Palu—XE6)

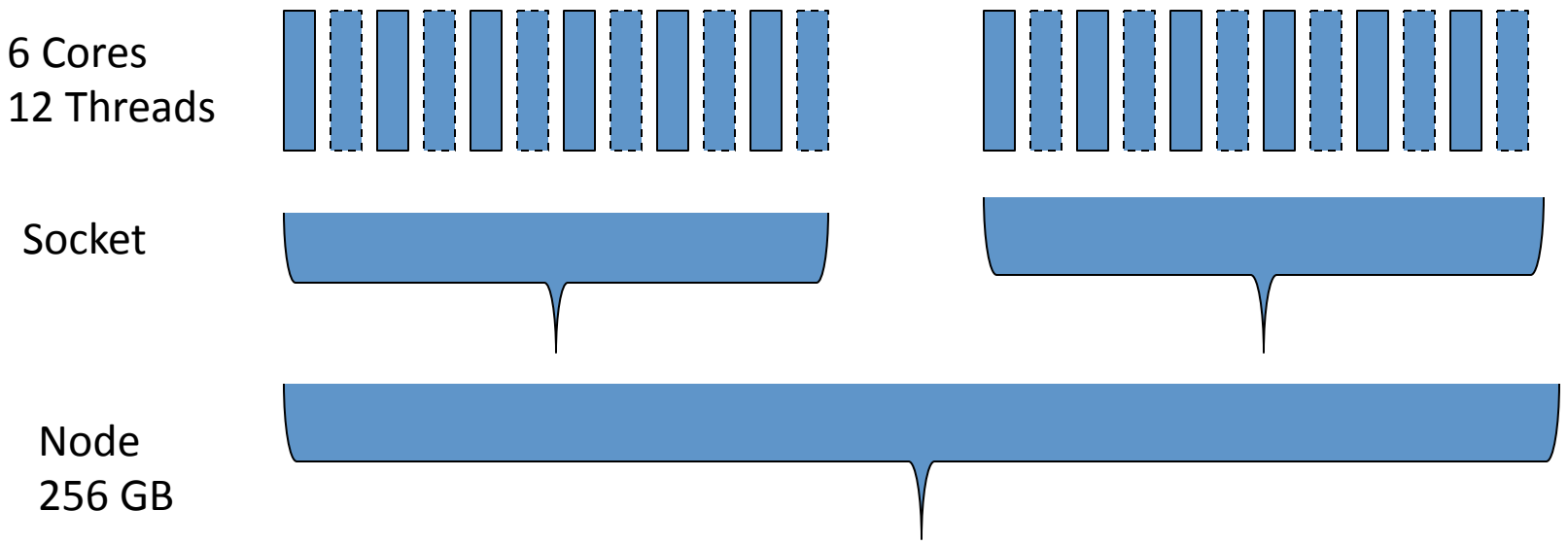
Cache levels (L1 & L2 not shared, L3 socket-level)

SSE Units (4 FLOP / clock cycle)



Xeon Based Cluster

Cache levels (L1, L2 & L3)
SSE Units (4 FLOP / clock cycle)
Hyper-threading



Speedup and Parallel Efficiency

- Amdahl's law
  - P (fraction of parallelized region)
  - S (speedup of parallelized region)

$$Speedup_{overall} = \frac{1}{(1-P) + \frac{P}{S}}$$

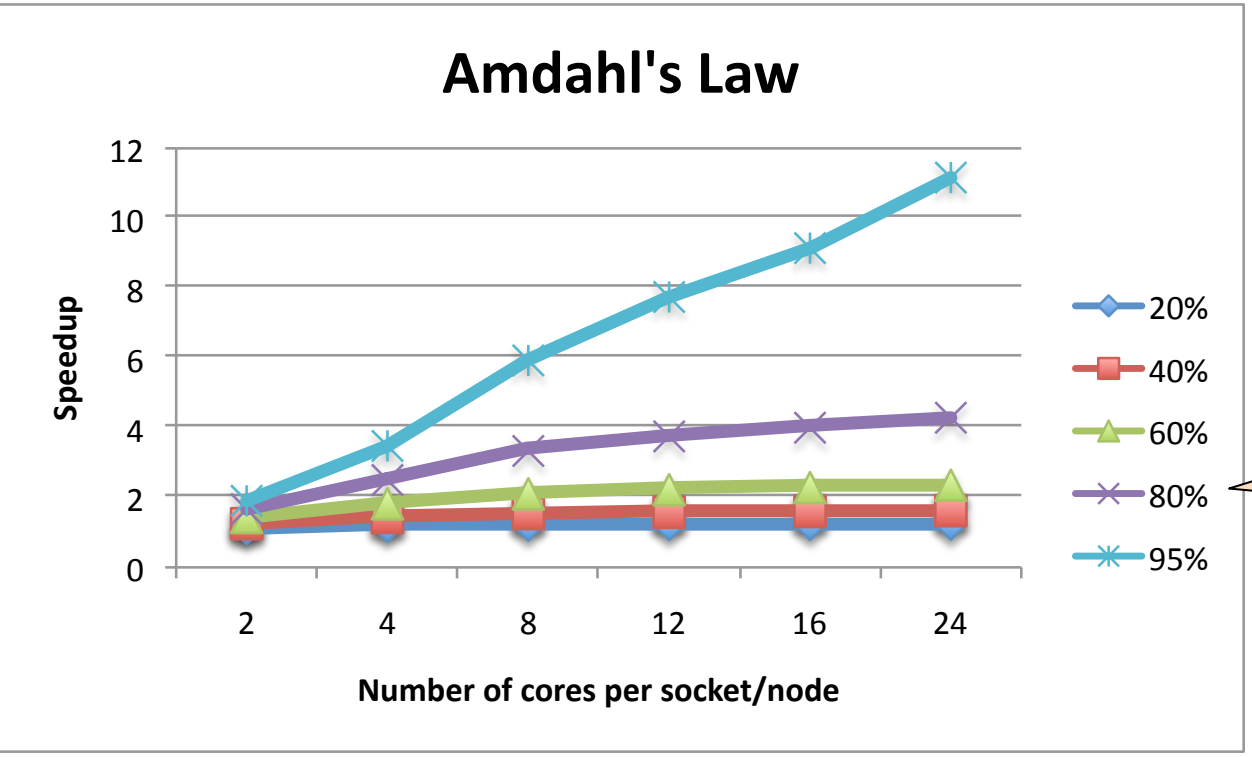
- Parallel efficiency
  - How efficiently resources are used

$$Efficiency_{parallel} = \frac{Speedup_{overall}}{Number_{resources}}$$

- Note: no overhead, contentions and load balancing issues considered

Threading (OpenMP) and Multi-cores (I)

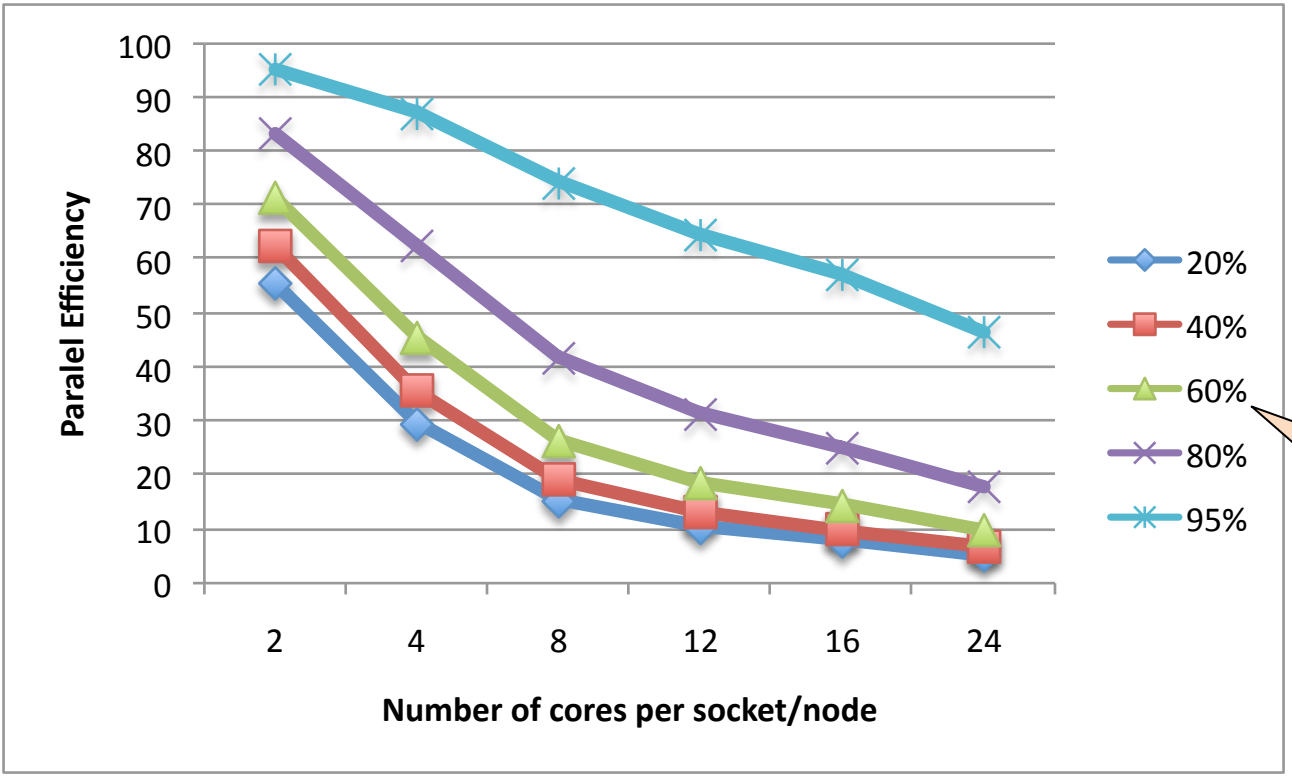
- Number of cores per socket/node growing
- Static thread/process/MPI binding on supercomputers



Percentage of parallel code (considering zero overhead)

Threading (OpenMP) and Multi-cores (II)

- Number of cores per socket/node growing
- Static thread/process/MPI binding on supercomputers



Percentage of parallel code (considering zero overhead)

Recap

- Peak performance attributes of emerging microprocessor
  - Cores, cores, & more
  - Vector units with increasing widths
  - Hardware assisted threading (hyper-threading)

- Multi-threading for:
  - Latency hiding
  - Task parallelism (computation and communication)
  - Dynamic load balancing
  - Address MPI decomposition limits (memory and task)

PART II—IDENTIFICATION OF SCALING AND PERFORMANCE BOTTLENECKS

Experiments (I)

- Systems
  - Cray XT5 system (Rosa)
  - Cray XE6 system (Palu)
  - An InfiniBand cluster

Results and analysis should be reproducible on systems with similar configurations

- Software
  - Compilers: Cray, PGI, Pathscale, GNU & Intel
  - MPI libraries: Vendor and MVAPICH2 (vendor MPI highly efficient and multi-core aware)
  - Performance tools: CrayPAT and TAU

Experiments (II)

- Software contd.
  - Runtime control
    - MPI task mapping using ALPS
    - Thread mapping using taskset on Xeon

Results and analysis should be reproducible on systems with similar configurations

- NAS Parallel Benchmarks suite v 3.3.1
  - Available from http://www.nas.nasa.gov/Resources/Software/npb.html
  - MPI “only” kernels and simulated applications
  - OpenMP “only” kernels and simulated applications
  - MPI+OpenMP simulated applications

Target Systems (Cray MPPs)

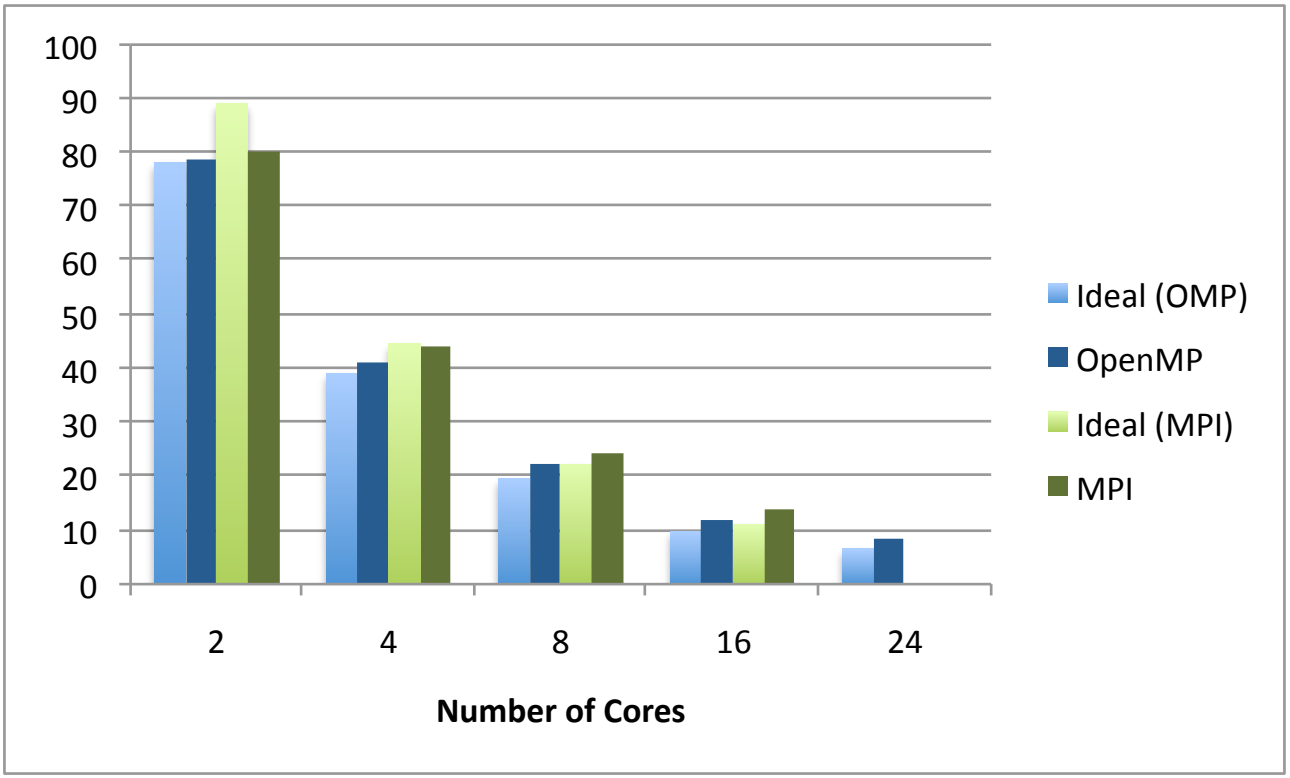
	Rosa	Palu
Product name	Cray XT5	Cray XE6
Node	Dual-socket 6-cores 2.6 GHz AMD Istanbul	Dual-socket 12-cores 2.1 GHz AMD Magny-cours
Cores per node	12	24
Node memory	16 GB	32 GB
Memory bandwidth/ node	25.6 GB/s	85.3 GB/s
Cache levels	64 KB (data)/ 512 KB/ 6 MB	64 KB (data)/ 512 KB/ 12 MB
Network	3-D Torus SeaStar2	2-D Torus GEMINI

Target System (Xeon IB cluster)

CPU Specifications	
Processor	Intel Xeon E7540
Nodes	2
Sockets per node	2
Cores per socket	6
Threads per socket	12
L1 cache	288 KB
L2 cache	1.5 MB
L3 cache	18 MB
Memory	256 GB
Network	DDR

Comparison using Vendor MPI

- Conjugate gradient kernel

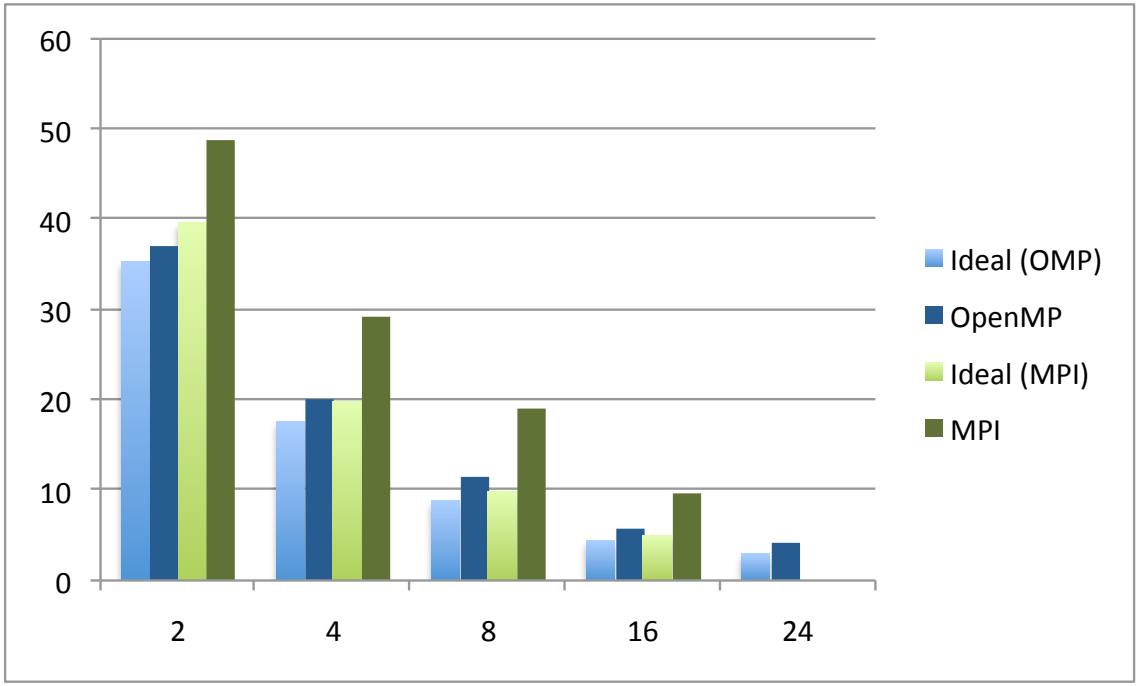


MPI (16) = ~14 sec
OpenMP (24) = ~ 8 sec

CG Code Modification Snapshots

Serial	MPI	OMP
do j=1, lastcol-firstcol+1 rho = rho + r(j)*r(j) enddo	sum = 0.0d0 do j=1, lastcol-firstcol+1 sum = sum + r(j)*r(j) enddo [followed by MPI send/ recv]	**!$omp do reduction(+:sum)** do j=1, lastcol-firstcol+1 suml = x(j) - r(j) sum = sum + suml*suml enddo **!$omp end do nowait**

Comparison using vendor MPI



- Fast Fourier Transform (FFT) kernel

MPI (16) = ~10 sec
OpenMP (24) = ~ 4 sec

FT Code Modification Snapshots

MPI

```
do k = 1, d3
      do jj = 0, d2 - fftblock, fftblock
        do j = 1, fftblock
          do i = 1, d1
            y(j,i,1) = x(i,j+jj,k)
          enddo
        enddo

        call cfftz (is, logd1, d1, y, y(1,1,2))

        do j = 1, fftblock
          do i = 1, d1
            xout(i,j+jj,k) = y(j,i,1)
          enddo
        enddo

      enddo
enddo
```

OMP

```
!$omp parallel do default(shared) private
(i,j,k,jj,y1,y2)
!$omp&  shared(is,logd1,d1)
    do k = 1, d3
      do jj = 0, d2 - fftblock, fftblock
        do j = 1, fftblock
          do i = 1, d1
            y1(j,i) = x(i,j+jj,k)
          enddo
        enddo
        call cfftz (is, logd1, d1, y1, y2)
        do j = 1, fftblock
          do i = 1, d1
            xout(i,j+jj,k) = y1(j,i)
          enddo
        enddo
      enddo
    enddo
```

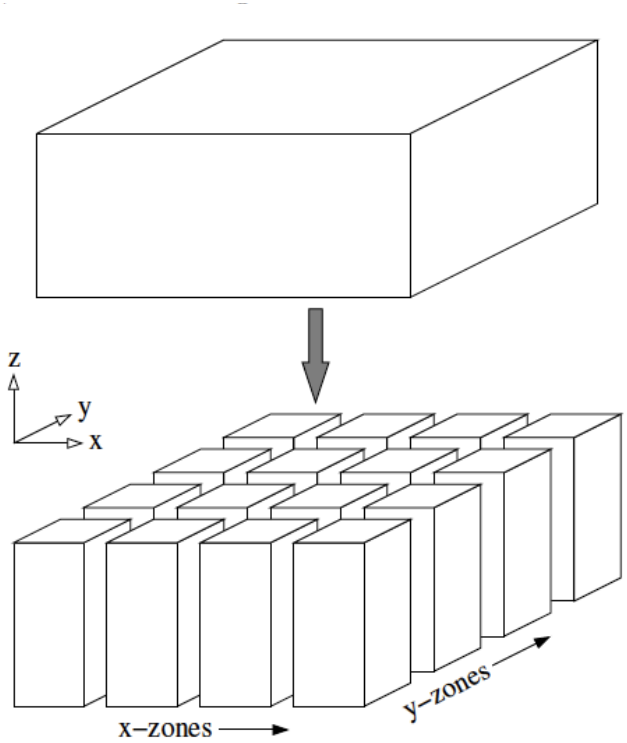
OpenMP Overhead

- Compare OpenMP single-threaded (OMP_NUM_THREAD=1) with serial version

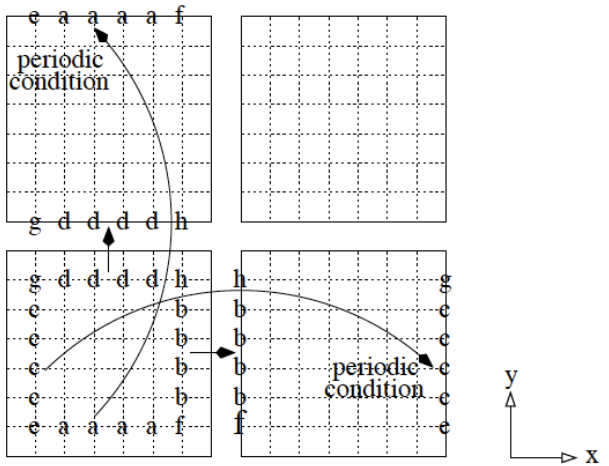
	CG	FFT
Serial	194.16	75.5
OpenMP (1)	198.4	76.54

- Likewise MPI_Init_thread (variants SINGLE, FUNNELED, SERIAL, MULTIPLE) did not add any noticeable overheads

Hybrid Implementations



- Number of zones fixed for a problem size<=max number of MPI tasks
- Number of threads could vary



Hybrid Results

- Lower-Upper Symmetric Gauss-Seidel (LU) application
  - Zones (4 x 4)
  - Problem size (304 x 208 x 17)

	2	4	8	16
1			89.4	45.18
2		49.76	35.97	
4		26.77		
6	34.85	17.46		
12	22.52			

MPI tasks

Threads

Same number of cores??

PART III—TUNING AND OPTIMIZATION ON TARGETED PLATFORMS

Reminder

- Target systems (multi-chip, multi-core)
  - Cray XT5 system (Rosa)
    - 12 cores per node (AMD hex-cores Istanbul)
    - 6-cores per socket
  - Cray XE6 system
    - 24 cores per node (AMD 12-cores Magny-cours)
    - 12 cores per socket
    - 6 cores per NUMA partition
  - An InfiniBand cluster
    - 12 core hyper-threaded (24 threads) per node (Intel Xeon)
    - 6 cores (12 hyper threads) per socket

Micro-processor Optimization and OpenMP Scaling

Serial
194.16

Serial
133.73

	PGI	PathScale	Cray	GNU
1	198.4	197.82	134.33	196.76
2	80.28	79.59	69.36	81.36
4	43.58	43.14	38.39	44.14
8	25.72	25.25	23.43	25.81

Breakdown of Runtime

Cray

```
 Time % |     Time |     Calls |Group
        |          |           | Function

 100.0% | 37.188682 | 3412039.0 |Total
|-----------------------------------------
| 100.0% | 37.188682 | 3412039.0 |USER
||----------------------------------------
||  77.2% | 28.695187 |      1.0 |cg_
||  21.0% |  7.793220 |      1.0 |sparse_
||   1.8% |  0.675186 |      1.0 |makea_
||   0.1% |  0.023118 | 3412017.0 |randlc_
||   0.0% |  0.001903 |      1.0 |print_results_
||   0.0% |  0.000041 |      1.0 |exit
||   0.0% |  0.000010 |      2.0 |timer_stop_
||   0.0% |  0.000009 |      4.0 |wtime_
||   0.0% |  0.000004 |      4.0 |elapsed_time_
.....
```

PGI

```
 Time % |     Time |     Calls |Group
        |          |           | Function

 100.0% | 50.391587 | 5268064.0 |Total
|-----------------------------------------
| 100.0% | 50.391587 | 5268064.0 |USER
||----------------------------------------
||  83.1% | 41.859248 |     16.0 |conj_grad_
||  14.4% |  7.231632 |      1.0 |sparse_
||   2.0% |  0.988592 |  75000.0 |sprnvc_
||   0.3% |  0.175017 | 3412017.0 |randlc_
||   0.1% |  0.064482 | 1706008.0 |icnvrt_
||   0.1% |  0.037387 |      1.0 |makea_
||   0.1% |  0.028324 |      1.0 |MAIN_
||   0.0% |  0.004952 |  75000.0 |vecset_
||   0.0% |  0.001881 |      1.0 |print_results_
||   0.0% |  0.000041 |      1.0 |exit
.....
```

Quiz

- What is going on here?
  - Same code
  - Similar compiler flags (-O3 for Cray –O3 –fastsse for PGI)
  - Same tool (CrayPAT)

Compiler Optimization (I)

- 238. w VpAr4 I--> call conj_grad (colidx,

Primary Loop Type	Modifiers
A - Pattern matched	a - vector atomic memory operation
	b - blocked
C - Collapsed	c - conditional and/or computed
D - Deleted	f - fused
E - Cloned	
I - Inlined	i - interchanged
M - Multithreaded	m - partitioned
P - Parallel	p - partial
R - Redundant	r - unrolled
	s - shortloop
V - Vectorized	t - array syntax temp used
	w - unwound

Compiler Optimization (II)

- What Cray –O3 translates to:

Options : -O cache2,fp2,scalar3,thread2,vector3,modinline,ipa3,noaggress
-O noautothread,nodwarf,fusion2,nomsgs,negmsgs,omp,nooverindex,pattern
-O shortcircuit2,unroll2,nozeroinc

- Do not need to compile the entire application for generating compiler listings

Mapping and Control of Tasks and Threads

- Useful commands
  - On Cray systems
    - Check man pages for ALPS
      - aprun –n (mpi tasks) –N (tasks per node) –S (tasks per socket) –d (thread depth) …
      - display_affinity(rank) function
  - On AMD systems
    - Check man pages for numactl
  - On Xeon systems
    - Check man pages for taskset

Remember:

All cores on a node share memory

Cores on a socket share L3 cache

Extra threads can help!!

Finding MPI Task and OMP Thread Affinity

```
/* Borrowed from util-linux-2.13-pre7/schedutils/taskset.c */
static char *cpuset_to_cstr(cpu_set_t *mask, char *str)
{
  char *ptr = str;
  int i, j, entry_made = 0;
  for (i = 0; i < CPU_SETSIZE; i++) {
    if (CPU_ISSET(i, mask)) {
      int run = 0;
      entry_made = 1;
      for (j = i + 1; j < CPU_SETSIZE; j++) {
        if (CPU_ISSET(j, mask)) run++;
        else break;
      }
      if (!run)
        sprintf(ptr, "%d,", i);
      else if (run == 1) {
        sprintf(ptr, "%d,%d,", i, i + 1);
        i++;
      } else {
        sprintf(ptr, "%d-%d,", i, i + run);
        i += run;
      }
      while (*ptr != 0) ptr++;
    }
  }
  ptr -= entry_made;
  *ptr = 0;
  return(str);
}
```

```
int main(int argc, char *argv[])
{
  int rank, thread;
  cpu_set_t coremask;
  char clbuf[7 * CPU_SETSIZE], hnbuf[64];

  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  memset(clbuf, 0, sizeof(clbuf));
  memset(hnbuf, 0, sizeof(hnbuf));
  (void)gethostname(hnbuf, sizeof(hnbuf));
  #pragma omp parallel private(thread, coremask, clbuf)
  {
    thread = omp_get_thread_num();
    (void)sched_getaffinity(0, sizeof(coremask), &coremask);
    cpuset_to_cstr(&coremask, clbuf);
    #pragma omp barrier
    printf("Hello from rank %d, thread %d, on %s.
    (core affinity = %s)\n", rank, thread, hnbuf, clbuf);
  }
  MPI_Finalize();
  return(0);
}
```

Sample Outputs (1)—4 MPI tasks & 4 OMP Threads One MPI Task Per Node (-N 1)

```
> export OMP_NUM_THREADS=4
> aprun -n 4 -N 1 -d 4 ./xthi
Hello from rank 0, thread 3, on nid00171. (core affinity = 3)
Hello from rank 0, thread 0, on nid00171. (core affinity = 0)
Hello from rank 0, thread 2, on nid00171. (core affinity = 2)
Hello from rank 0, thread 1, on nid00171. (core affinity = 1)
Hello from rank 1, thread 3, on nid00172. (core affinity = 3)
Hello from rank 2, thread 0, on nid00173. (core affinity = 0)
Hello from rank 3, thread 3, on nid00178. (core affinity = 3)
Hello from rank 3, thread 0, on nid00178. (core affinity = 0)
Hello from rank 2, thread 3, on nid00173. (core affinity = 3)
Hello from rank 1, thread 0, on nid00172. (core affinity = 0)
Hello from rank 3, thread 1, on nid00178. (core affinity = 1)
Hello from rank 2, thread 1, on nid00173. (core affinity = 1)
Hello from rank 2, thread 2, on nid00173. (core affinity = 2)
Hello from rank 1, thread 1, on nid00172. (core affinity = 1)
Hello from rank 3, thread 2, on nid00178. (core affinity = 2)
Hello from rank 1, thread 2, on nid00172. (core affinity = 2)
```

Sample Outputs ()—4 MPI tasks & 4 OMP Threads 2 MPI Tasks Per Node, One Per NUMA Socket

```
> aprun -n 4 -N 2 -d 4 -S 1 ./xthi
Hello from rank 0, thread 3, on nid00171. (core affinity = 3)
Hello from rank 0, thread 0, on nid00171. (core affinity = 0)
Hello from rank 0, thread 2, on nid00171. (core affinity = 2)
Hello from rank 1, thread 0, on nid00171. (core affinity = 6)
Hello from rank 1, thread 1, on nid00171. (core affinity = 7)
Hello from rank 1, thread 2, on nid00171. (core affinity = 8)
Hello from rank 1, thread 3, on nid00171. (core affinity = 9)
Hello from rank 0, thread 1, on nid00171. (core affinity = 1)
Hello from rank 2, thread 3, on nid00172. (core affinity = 3)
Hello from rank 2, thread 0, on nid00172. (core affinity = 0)
Hello from rank 3, thread 3, on nid00172. (core affinity = 9)
Hello from rank 2, thread 2, on nid00172. (core affinity = 2)
Hello from rank 3, thread 1, on nid00172. (core affinity = 7)
Hello from rank 2, thread 1, on nid00172. (core affinity = 1)
Hello from rank 3, thread 2, on nid00172. (core affinity = 8)
Hello from rank 3, thread 0, on nid00172. (core affinity = 6)
```

Sample Outputs (3)—4 MPI tasks & 6 OMP Threads
1 MPI Task Per Node, One Task Per NUMA Socket

```
> aprun -n 4 -N 1 -d 6 -S 1 ./xthi
Hello from rank 0, thread 5, on nid00171. (core affinity = 5)
Hello from rank 0, thread 1, on nid00171. (core affinity = 1)
Hello from rank 0, thread 2, on nid00171. (core affinity = 2)
Hello from rank 0, thread 0, on nid00171. (core affinity = 0)
Hello from rank 0, thread 4, on nid00171. (core affinity = 4)
Hello from rank 0, thread 3, on nid00171. (core affinity = 3)
Hello from rank 1, thread 5, on nid00172. (core affinity = 5)
Hello from rank 1, thread 2, on nid00172. (core affinity = 2)
Hello from rank 2, thread 5, on nid00173. (core affinity = 5)
Hello from rank 3, thread 5, on nid00178. (core affinity = 5)
Hello from rank 3, thread 0, on nid00178. (core affinity = 0)
Hello from rank 3, thread 1, on nid00178. (core affinity = 1)
Hello from rank 3, thread 3, on nid00178. (core affinity = 3)
Hello from rank 3, thread 2, on nid00178. (core affinity = 2)
Hello from rank 3, thread 4, on nid00178. (core affinity = 4)
Hello from rank 2, thread 0, on nid00173. (core affinity = 0)
Hello from rank 1, thread 3, on nid00172. (core affinity = 3)
Hello from rank 2, thread 1, on nid00173. (core affinity = 1)
Hello from rank 1, thread 0, on nid00172. (core affinity = 0)
Hello from rank 2, thread 3, on nid00173. (core affinity = 3)
Hello from rank 2, thread 2, on nid00173. (core affinity = 2)
Hello from rank 2, thread 4, on nid00173. (core affinity = 4)
Hello from rank 1, thread 4, on nid00172. (core affinity = 4)
Hello from rank 1, thread 1, on nid00172. (core affinity = 1)
```

Sample Outputs (4)—2 MPI tasks & 6 OMP Threads
2 MPI Tasks Per Node, One Task Per NUMA Socket

```
> export OMP_NUM_THREADS=6
> aprun -n 2 -N 2 -d 6 -S 1 ./xthi
Hello from rank 0, thread 0, on nid00171. (core affinity = 0)
Hello from rank 0, thread 5, on nid00171. (core affinity = 5)
Hello from rank 1, thread 0, on nid00171. (core affinity = 6)
Hello from rank 0, thread 4, on nid00171. (core affinity = 4)
Hello from rank 1, thread 5, on nid00171. (core affinity = 11)
Hello from rank 0, thread 1, on nid00171. (core affinity = 1)
Hello from rank 1, thread 3, on nid00171. (core affinity = 9)
Hello from rank 0, thread 3, on nid00171. (core affinity = 3)
Hello from rank 1, thread 4, on nid00171. (core affinity = 10)
Hello from rank 0, thread 2, on nid00171. (core affinity = 2)
Hello from rank 1, thread 2, on nid00171. (core affinity = 8)
Hello from rank 1, thread 1, on nid00171. (core affinity = 7)
```

Finding Thread Affinity on Rosa

```
(example by Tim Robinson, CSCS)

program hello
use mpi
integer rank, size, ierror
call MPI_INIT(ierror)
call MPI_COMM_SIZE(MPI_COMM_WORLD, size, ierro
call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierro
print*, 'rnk is', rank, ': Hello world'
call display_affinity(rank)
call MPI_FINALIZE(ierror)
end
```

```
ftn hello.f90 -L/apps/rosa/
affinity/1.0.1/pgi_9.0.3 -
laffinity
```

```
> export OMP_NUM_THREADS=6
> aprun -n 4 -N 2 -S 1 -d 6 ./a.out
 rnk is            0 : Hello world
 rnk is            1 : Hello world
Rank 1, thread 0, on nid01844. (core affinity =6)
Rank 1, thread 2, on nid01844. (core affinity =8)
Rank 1, thread 5, on nid01844. (core affinity =11)
Rank 1, thread 1, on nid01844. (core affinity =7)
Rank 1, thread 4, on nid01844. (core affinity =10)
Rank 1, thread 3, on nid01844. (core affinity =9)
Rank 0, thread 0, on nid01844. (core affinity =0)
Rank 0, thread 2, on nid01844. (core affinity =2)
Rank 0, thread 3, on nid01844. (core affinity =3)
Rank 0, thread 4, on nid01844. (core affinity =4)
Rank 0, thread 5, on nid01844. (core affinity =5)
Rank 0, thread 1, on nid01844. (core affinity =1)
 rnk is            2 : Hello world
 rnk is            3 : Hello world
Rank 3, thread 0, on nid01332. (core affinity =6)
Rank 3, thread 4, on nid01332. (core affinity =10)
Rank 3, thread 2, on nid01332. (core affinity =8)
Rank 3, thread 5, on nid01332. (core affinity =11)
Rank 3, thread 1, on nid01332. (core affinity =7)
Rank 3, thread 3, on nid01332. (core affinity =9)
Rank 2, thread 0, on nid01332. (core affinity =0)
Rank 2, thread 2, on nid01332. (core affinity =2)
Rank 2, thread 1, on nid01332. (core affinity =1)
Rank 2, thread 3, on nid01332. (core affinity =3)
Rank 2, thread 4, on nid01332. (core affinity =4)
Rank 2, thread 5, on nid01332. (core affinity =5)
```

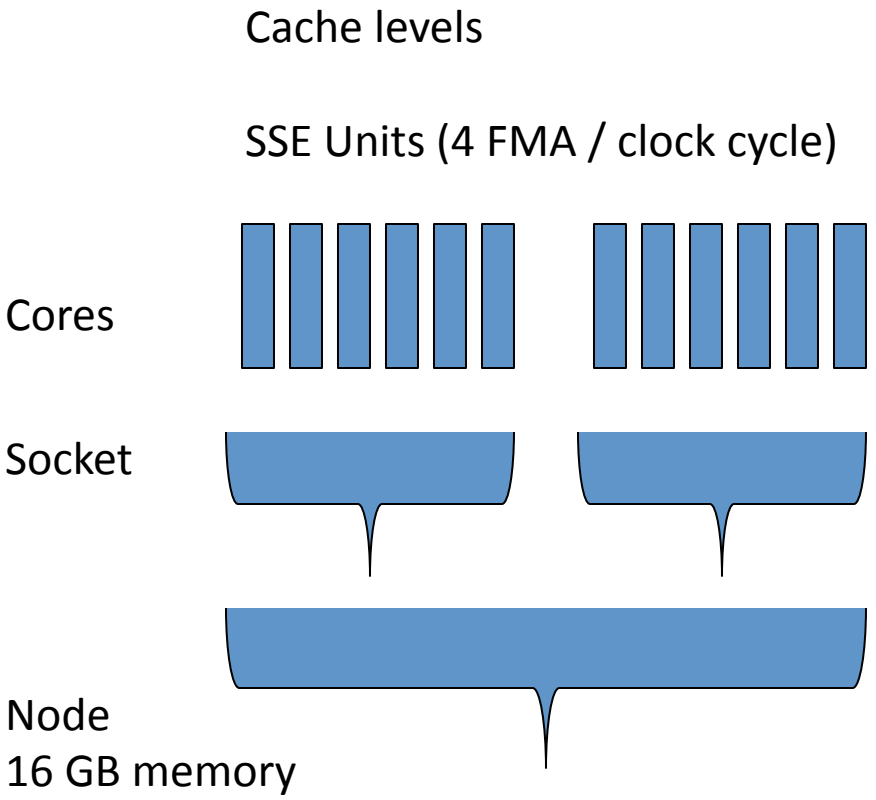
Impact of Mapping on Multi-socket Core

	2	4
CG	33.78	19.24
CG (map)	31.86	17.13
FFT	40.49	23.31
FFT (map)	38.77	20.84

Remember Affinities and Bindings:

Task

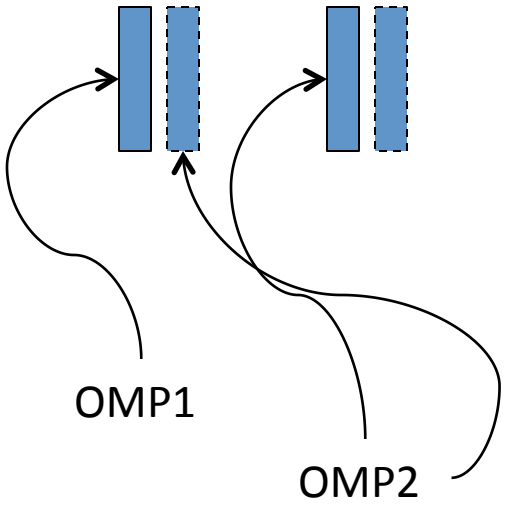
Memory



Hyper-threading on Xeon

	Non HT	HT
4	38.24	70.86
8	19.83	36.46
12	13.5	25.04
16		23.57
24		12.92

	bt.C	is.C	ep.C
12	37.54	2.51	17.05
24	39.02	2.02	14.44



Performance Measurement and Analysis

- On Cray systems for Hybrid applications:
  - CrayPAT
  - TAU

- Using CrayPAT
  - Load module
  - Build application
  - Instrument application using mpi (-g mpi) and openmp (-g omp)
  - Run application
  - Analyze results (different pat_report flags or visual analyzer)

CrayPAT Output (I)

```
Time % |     Time |  Calls |Group
       |          |        | Function
       |          |        |  Thread

 100.0% | 20.517937 | 8999.0 |Total
|-----------------------------------
| 100.0% | 20.514984 | 8295.0 |USER
||----------------------------------
||  93.3% | 19.152761 | 7800.0 |cg_.LOOP@li.319
|||---------------------------------
3|   7.8% | 19.152761 | 7800.0 |thread.11
3|   7.8% | 19.152623 | 7800.0 |thread.3
3|   7.8% | 19.152475 | 7800.0 |thread.2
3|   7.8% | 19.152221 | 7800.0 |thread.0
3|   7.8% | 19.152073 | 7800.0 |thread.5
3|   7.8% | 19.152011 | 7800.0 |thread.9
3|   7.8% | 19.151967 | 7800.0 |thread.1
3|   7.8% | 19.151792 | 7800.0 |thread.8
3|   7.8% | 19.151702 | 7800.0 |thread.7
3|   7.8% | 19.151547 | 7800.0 |thread.6
3|   7.8% | 19.151255 | 7800.0 |thread.4
3|   7.8% | 19.150800 | 7800.0 |thread.10
```

```
Time % |     Time |  Calls |Group
       |          |        | Function
       |          |        |  Caller
       |          |        |   Thread='HIDE'

 100.0% | 20.517937 | 8999.0 |Total
|-----------------------------------------
| 100.0% | 20.514984 | 8295.0 |USER
||----------------------------------------
||  93.3% | 19.152761 | 7800.0 |cg_.LOOP@li.319
3|       |           |        | (N/A):(N/A):line.0
||   4.7% |  0.954759 |    1.0 |cg_.REGION@li.186
3|       |           |        | (N/A):(N/A):line.0
||   1.3% |  0.257450 |  104.0 |cg_.LOOP@li.245
3|       |           |        | (N/A):(N/A):line.0
||   0.4% |  0.089411 |    1.0 |sparse_.LOOP@li.1009
3|       |           |        | (N/A):(N/A):line.0
||   0.2% |  0.033126 |    1.0 |cg_
3|       |           |        | (N/A):(N/A):line.0
||   0.1% |  0.010817 |   75.0 |cg_.REGION@li.319
```

Code not instrumented correctly

CrayPAT Output (II)

```
 Time % |     Time |Imb. Time |   Imb. |   Calls |Group
        |          | Time % |        |         | Function
        |          |        |        |         | Thread='HIDE'

 100.0% | 9.365307 |      -- |     -- | 31879.0 |Total
|-----------------------------------------------------------------
|  98.6% | 9.229841 |      -- |     -- | 31532.0 |USER
||----------------------------------------------------------------
||  39.4% | 3.687824 | 0.146582 |   4.3% | 19360.0 |cfftz
||  25.4% | 2.377127 | 0.141248 |   6.5% |    20.0 |ft_.LOOP@li.137
||  10.1% | 0.949889 | 0.040375 |   4.6% |    22.0 |cffts3_.LOOP@li.
631
||   7.4% | 0.691512 | 0.029757 |   4.7% |    22.0 |      OP@li.586
||   7.2% | 0.675323 | 0.033533 |   5.4% |         |      OP@li.107
||   6.4% | 0.595786 | 0.021274 |   3.7% |         |cffts1_.LOOP@li.540
||   1.1% | 0.098958 | 0.007064 |   7.1% |  1264.0 |vranlc_
||
================================================================
======
|   1.4% |          |          |     -- |     -- |   336.0 |OMP
||----------------------------------------------------------------
||   1.1% | 0.099397 | 0.091114 | 100.0% |     1.0 |ft_.REGION@li.107
(ovhd)
|
================================================================
=======
```

```
 Time % |     Time |   Calls |Group
        |          |         | Thread

 100.0% | 9.325101 | 31879.0 |Total
|-----------------------------------
|  98.5% | 9.189635 | 31532.0 |USER
||----------------------------------
||   8.6% | 9.189635 | 31532.0 |thread.0
||   8.6% | 9.169371 | 30847.0 |thread.4
||   8.6% | 9.145816 | 30847.0 |thread.2
||   8.5% | 9.133687 | 30847.0 |thread.1
||   8.5% | 9.132083 | 30847.0 |thread.5
||   8.5% | 9.128529 | 30847.0 |thread.3
||   8.4% | 8.930311 | 30847.0 |thread.10
||   8.3% | 8.924085 | 30847.0 |thread.8
||   8.3% | 8.917861 | 30847.0 |thread.7
||   8.3% | 8.917753 | 30847.0 |thread.6
||   8.3% | 8.907880 | 30847.0 |thread.9
||   5.5% | 5.890968 | 19711.0 |thread.11
```

Quiz: what is this type of info called?

Quiz: How Do I Generate Different Listings

Available option values are in left column, a prefix can be specified:

ct	-O calltree
heap	-O heap_program,heap_hiwater,heap_leaks
io	-O read_stats,write_stats
lb	-O load_balance
load_balance	-O lb_program,lb_group,lb_function
mpi	-O mpi_callers

Options for related tables not shown by default:

-O profile_pe.th	-O callers
-O profile_th_pe	-O callers+src
-O profile+src	-O calltree
-O load_balance	-O calltree+src

There are more …..

Key Considerations

- OpenMP version of the code should retain micro-processor optimization—check compiler listings
  - SSE instructions
  - Loop level optimization
  - Inlining and IPA (where applicable)

- OpenMP single thread should not be considerably slower than the serial version

Main Advantages of using OpenMP

- Typically can have as many threads as number of cores
- More memory per MPI task
- Dynamic load balancing opportunities using different thread scheduling schemes
- Users can dynamically choose fewer threads to boost performance

Myths and Realities

- OpenMP is mostly/always slow
  - Can be faster—depends on the implementation and tuning effort
- OpenMP adds a lot of overhead
  - Check against the serial version on a single core—shouldn't be more than 15% or review your implementation

Myths and Realities (II)

- OpenMP does not scale
  - Depends on fraction of OpenMPized code and implementation techniques

- MPI threaded versions are slow
  - Measured data on Cray platforms (using MPI_Init_thread variants) did not support this claim

Future of MPI+OpenMP

- MPI3 standards committee
  - Documents & minutes of meetings available
  - http://meetings.mpi-forum.org/MPI_3.0_main_page.php

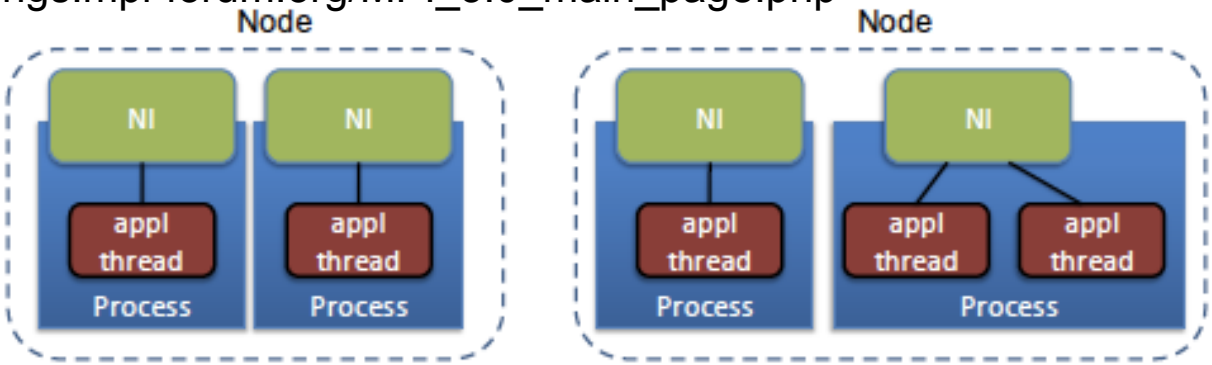


Figure 2: MPI Current Design

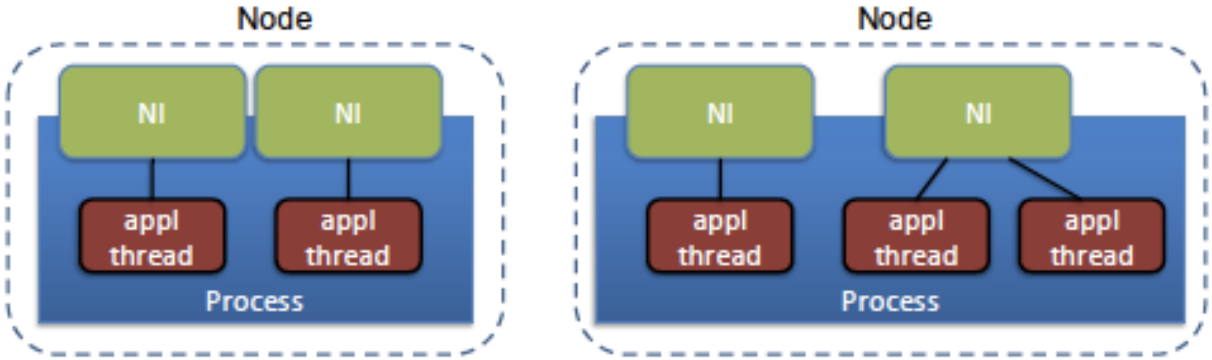


Figure 3: MPI Proposed Design

Directives Based Programming for GPU Devices

- PGI accelerator directives
  - CSCS course on March 29-31
- Cray GPU directives based programming
  - Evaluation under way
- Standardization effort underway within OpenMP consortium
  - Potential for high-productivity and high-performance portable programming on heterogeneous devices

References (links to specifications, tutorials, etc.)

- http://www.mpi-forum.org/
- https://svn.mpi-forum.org/trac/mpi-forum-web/wiki
- http://openmp.org/wp/
- http://www.compunity.org/
- http://www.cs.uoregon.edu/research/tau/home.php
- http://docs.cray.com/
- http://www.nas.nasa.gov/Resources/Software/npb.html
- R. Rabenseifner, *et. al.* "Hybrid MPI/OpenMP Parallel Programming on Clusters of Multi-Core SMP Nodes", 2009 Parallel, Distributed and Network-based Processing