Best Practices for Hybrid OpenMP/MPI Programming on Hopper.
The Cray Center of Excellence: Performance Optimization for the Multicore Era
GTC, Gamess, fvCAM and PARATEC

Nicholas J Wright, Karl Fuerlinger, Hongzhang Shan,
Tony Drummond, Andrew Canning, and John Shalf
NERSC/LBNL
Stephane Ethier
Princeton Plasma Physics Lab
Nathan Wichmann, Marcus Wagner, Sarah Anderson, Ryan Olsen, and Mike Aamodt
Cray Inc
The Multicore era

- Moore’s Law continues

- Traditional sources of performance improvement ending
  - Old Trend: double clock frequency every 18th months
  - New Trend: Double # cores every 18 months

- Power Limits Drive a number of Broader Technology Trends
  - Number Cores $\uparrow$
  - Memory Capacity per core $\downarrow$
  - Memory Bandwidth per FLOP $\downarrow$
  - Network Bandwidth per FLOP $\downarrow$

Figure courtesy of Kunle Olukotun, Lance Hammond, Herb Sutter, and Burton Smith
The Multicore era

• Moore’s Law continues

• Traditional sources of performance improvement ending
  – Old Trend: double clock frequency every 18th months
  – New Trend: Double # cores every 18 months

• Implication for NERSC users
  – 3x increase in system performance with no per-core performance improvement
  – 12x more cores in NERSC-6 (hopper) than NERSC-5 (franklin) (4 cores to 24 cores)
  – Same or lower memory capacity per core on compute nodes

• Flat MPI-only model for parallelism will not scale
  – Need to transition to new model that can sustain massive growth in parallelism
  – Hopper changes are first step in a long-term technology trend
  – NERSC needs to take pro-active role in guiding transition of user community
Long-Term Concerns for NERSC Users

- Franklin (N5) 19 TF Sustained, 101 TF Peak
- Franklin (N5) + QC 36 TF Sustained, 352 TF Peak
- Top500
- COTS/MPP + MPI
- COTS/MPP + MPI (+ OpenMP)
- Hopper (N6) >1 PF Peak
- NERSC-7 10 PF Peak
- NERSC-8 100 PF Peak
- NERSC-9 1 EF Peak
- GPU CUDA/OpenCL or Manycore BG/Q, R
- Exascale + ????


Peak Teraflop/s

10^7 10^6 10^5 10^4 10^3 10^2 10
NERSC/Cray “Programming Models Center of Excellence” combines:
• Berkeley Lab strength in advanced programming models, multicore tuning, and application benchmarking
• Cray strength in advanced programming models, optimizing compilers, and benchmarking

Immediate question: What is the best way to use cores in N6 (Hopper) node?
• Flat MPI - Today’s preferred mode of operation
  – Model has diverged from reality (the machine is NOT flat)
  – 4 - 8 cores? ✔ 128 - 1024 cores? ✗
• MPI + OpenMP
• MPI + pthreads
• MPI + PGAS
• PGAS, CUDA, OPENCL, ....

Multicore Era: Massive on-chip concurrency necessary for reasonable power use
NERSC COE: Project Plan

- **Phase 1: Prepare users for Hopper**
  - NERSC-6 application benchmarks provide representative set of NERSC workload and broad cross-section of algorithms
  - User hybrid OpenMP/MPI model because it is most mature
  - Analyze performance of hybrid applications
  - Work with USG to create training materials for Hopper users

- **Phase 2: Prepare users for next decade**
  - Evaluate advanced programming models
  - Identify durable approach for programming on path to exascale
AMD – Magny-Cours

G34 Socket
“Magny-Cours” utilizes a Directly Connected MCM

Package has 12 cores, 4 HT ports, & 4 memory channels

Die (Node) has 6 cores, 4 HT ports & 2 memory channels

DDR3 Memory Channel

x16 cHT
x8 cHT
x16 (NC)
x16 ncHT
• “Deeper” Memory Hierarchy
  – NUMA: Non-Uniform Memory Architecture
  – All memory is transparently accessible but...
  – Longer memory access time to “remote” memory
  – A process running on NUMA node 0 accessing NUMA node 1 memory can adversely affect performance.

2xDDR1333 channel
21.3 GB/s

3.2GHz x16 lane HT
12.8 GB/s bidirectional

3.2GHz x8 lane HT
6.4 GB/s bidirectional
What are the Basic Differences Between MPI and OpenMP?

**Message Passing Model**

- Program is a collection of processes.
  - Usually fixed at startup time
- Single thread of control plus private address space -- **NO shared data.**
- Processes communicate by explicit send/receive pairs
  - Coordination is implicit in every communication event.
- MPI is most important example.

**Shared Address Space Model**

- Program is a collection of threads.
  - Can be created dynamically.
- Threads have private variables and shared variables
- Threads communicate implicitly by writing and reading shared variables.
  - Threads coordinate by synchronizing on shared variables
- OpenMP is an example
Why does it matter? - NUMA mem latency

![Graph showing latency vs memory size for different nodes.](image)
Why does it matter? - NUMA mem bandwidth

Node 0 <= Node 0 .. 3

bw_mem -P 1 -N 5 1024M [benchmark]

Benchmark:
- bcopy
- bzero
- fcp
- frd
- fwr
- cp
- rdwr
- wr
- rd

Relative Bandwidth (%)
Double a[N], b[N], c[N];

......

#pragma omp parallel for
#endif
for (j=0; j< VectorSize; j++) {
    a[j] = 1.0; b[j] = 2.0; c[j] = 0.0;
}
#pragma omp parallel for
for (j=0; j< VectorSize; j++) {
    a[j]=b[j]+d*c[j];
}

...
Stream NUMA effects - Hopper
<table>
<thead>
<tr>
<th>Science areas</th>
<th>Dense linear algebra</th>
<th>Sparse linear algebra</th>
<th>Spectral Methods (FFT)s</th>
<th>Particle Methods</th>
<th>Structured Grids</th>
<th>Unstructured or AMR Grids</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accelerator Science</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>IMPACT-T</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>IMPACT-T</td>
<td>IMPACT-T</td>
<td></td>
</tr>
<tr>
<td>Astrophysics</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>MAESTRO</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MAESTRO</td>
<td></td>
</tr>
<tr>
<td>Chemistry</td>
<td>X</td>
<td>GAMESS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CAM</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>CAM</td>
<td></td>
</tr>
<tr>
<td>Climate</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CAM</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>CAM</td>
<td></td>
</tr>
<tr>
<td>Combustion</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MAESTRO</td>
<td>AMR Elliptic</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MAESTRO</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AMR Elliptic</td>
<td></td>
</tr>
<tr>
<td>Fusion</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>GTC</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>GTC</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>GTC</td>
<td></td>
</tr>
<tr>
<td>Lattice Gauge</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>MILC</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>MILC</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>MILC</td>
<td></td>
</tr>
<tr>
<td>Material Science</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MILC</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>MILC</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Hybrid MPI-OpenMP Programming

Benefits

+ Less Memory usage
+ Focus on # nodes (which is not increasing as fast) instead of # cores
+ Larger messages, less time in MPI
+ Attack different levels of parallelism than possible with MPI

Potential Pitfalls

- NUMA / Locality effects
- Synchronization overhead
- Inability to saturate network adaptor

Mitigations

- User training
- Code examples using real applications
- Hopper system configuration changes
- Feedback to Cray on compiler & system software development
Important to Set Expectations

• OpenMP + MPI unlikely to be faster than pure MPI - but it will almost certainly use less memory
• Very important to consider your overall performance
  – individual kernels maybe slower with OpenMP but the code overall maybe faster
• Sometimes it maybe better to leave cores idle
  – #1 Memory Capacity
  – #2 Memory Bandwidth
  – #3 Network Bandwidth
  – #4 Flops
Understanding Hybrid MPI/OPENMP Model

\[ T(N_{\text{MPI}}, N_{\text{OMP}}) = t(N_{\text{MPI}}) + t(N_{\text{OMP}}) + t(N_{\text{MPI}}, N_{\text{OMP}}) + t_{\text{serial}} \]

- **Serial**
  - count = \( \frac{G}{N_{\text{MPI}}} \)
  - Do i = 1, count

- **Parallel**
  - count = \( \frac{G}{N_{\text{OMP}}} \)
  - !$omp do private (i)
  - Do i = 1, G

- MPI
  - count = \( \frac{G}{N_{\text{OMP}} \times N_{\text{MPI}}} \)
  - !$omp do private (i)
  - Do i = 1, \( \frac{G}{N_{\text{MPI}}} \)

- **Serial**
  - count = G
  - Do i = 1, G
Breaking Down the Runtime - Tools

• IPM – Integrated Performance Monitoring
  http://ipm-hpc.sourceforge.net
  – Time in MPI, Messages sizes, Communication Patterns
  – Simple Interface to PAPI
  – OpenMP profiler module added

• OMPP – OpenMP Profiler
  http://www.cs.utk.edu/~karl/ompp.html
  – Time Spent in Openmp per region, Load imbalance, Overhead
  – Also Interfaces to PAPI
```
## IPMv0.xx##
#
# command : /tmp/work/nwright/for_nick/CAM_1.0/run/../../benchmark/bld/cam.ipm
# start   : Tue Jun 15 10:36:57 2010  host : nid21827
# stop    : Tue Jun 15 10:49:15 2010  wallclock : 737.20
# mpi_tasks : 20 on 20 nodes
# omp_thrs : 12
# mem [GB] : 0.00
#
# |        | total | avg | min | max |
# |------------------|------------------|
# wallclock       : 1473.19 | 736.91 | 736.85 | 737.20 |
# MPI             : 3471.63 | 173.58 | 138.00 | 212.08 |
# OMP             : 10476.12 | 523.81 | 488.26 | 548.34 |
# OMP idle        : 0.00  | 0.00  | 0.00  | 0.00  |
# %wall           : 23.56 | 18.73 | 28.78 |
# calls           : 7268732 | 363436 | 292369 | 411990 |
# mem [GB]        : 0.00  | 0.00  | 0.00  | 0.00  |
#
# |        | time | count | %wall |
# |------------------|------------------|
# OMP_PARALLEL    : 10476.12 | 4911120 | 71.08 |
# MPI_Waitall     : 1094.59  | 1789424 | 7.43  |
# MPI_Wait        : 546.18   | 1245742 | 3.71  |
# MPI_Alltoallv   : 501.70   | 19300   | 3.40  |
# MPI_Bcast       : 433.16   | 11980   | 2.94  |
# MPI_Bcast       : 275.12   | 30080   | 7.55  |
```
Talk Outline

- Gyrokinetic Toroidal Code (GTC)
- Parallel Total Energy Code (PARATEC)
- Finite Volume Community Atmosphere Model (fvCAM)
- General Atomic and Molecular Electronic Structure System (Gamess)
- Conclusions
- Next Steps
Gyrokinetic Toroidal Code (GTC)

- 3D Particle-in-cell (PIC)
- Used for simulations of non-linear gyrokinetic plasma microturbulence
- Paralleised with OpenMP and MPI.
- ~15K lines of Fortran 90
- OpenMP version 56 parallel regions/loops (almost all)
- 10 loops required different implementation for OpenMP version (~250 lines)
Particle-in-cell (PIC) method

- Particles sample distribution function (markers).
- The particles interact via a grid, on which the potential is calculated from deposited charges.

The PIC Steps
- “SCATTER”, or deposit, charges on the grid (nearest neighbors)
- Solve Poisson equation
- “GATHER” forces on each particle from potential
- Move particles (PUSH)
- Repeat…
Important Routines in GTC

Poisson – charge distribution ➔ Electric field
Charge – deposits charge on Grid
Smooth – smoothes charge on grid
Pusher – Moves the Ions/Electrons
Field – Calculates Forces due to Electric field
Shifter – Exchanges between MPI tasks
GTC MPI+OpenMP Performance

- **Pusher**
- **Shift**
- **Charge**
- **Poisson**

<table>
<thead>
<tr>
<th>OPENMP threads / MPI tasks</th>
<th>Time / Secs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1536</td>
</tr>
<tr>
<td>2</td>
<td>768</td>
</tr>
<tr>
<td>4</td>
<td>384</td>
</tr>
<tr>
<td>6</td>
<td>256</td>
</tr>
<tr>
<td>12</td>
<td>128</td>
</tr>
</tbody>
</table>

**Good**
<table>
<thead>
<tr>
<th>OPENMP threads / MPI tasks</th>
<th>Memory per node / GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>0.5</td>
</tr>
</tbody>
</table>

**GTC - Memory Usage**

- **OPENMP threads / MPI tasks**: The number of threads or tasks increases from 1 to 12, with corresponding memory usage in GB.
- **Memory per node / GB**: The memory usage decreases as the number of threads or tasks increases, indicating efficient resource management.
Small Test Case – 96 cores – Breakdown

The graph shows the breakdown of time (in seconds) for different processes across various numbers of OPENMP threads and MPI tasks. The processes are:
- poisson
- charge
- smooth
- pusher
- field
- shift

The X-axis represents the number of OPENMP threads and MPI tasks, ranging from 1 to 32, 24, 16, and 8, while the Y-axis represents the time in seconds, ranging from 0 to 200.

The data indicates that as the number of threads and tasks increases, the time required for each process decreases, suggesting better parallelization efficiency.
Small Test Case – 96 cores – Breakdown

- poisson
- charge
- smooth
- pusher
- field
- shift

<table>
<thead>
<tr>
<th>OPENMP threads / MPI tasks</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>6</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>96</td>
<td>180</td>
<td>80</td>
<td>60</td>
<td>40</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>48</td>
<td>140</td>
<td>70</td>
<td>50</td>
<td>30</td>
<td>15</td>
<td>7</td>
</tr>
<tr>
<td>32</td>
<td>100</td>
<td>50</td>
<td>30</td>
<td>20</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>24</td>
<td>60</td>
<td>30</td>
<td>20</td>
<td>10</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>16</td>
<td>40</td>
<td>20</td>
<td>10</td>
<td>5</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>20</td>
<td>10</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Small Test Case – 96 cores – Breakdown
Small Case - Performance Breakdown

![Graph showing performance breakdown with time on the y-axis and number of OPENMP threads on the x-axis. The graph includes lines for wallt, MPI time, Poisson, and Wallt-MPI-Poisson.]
GTC: Communication Analysis

![Graph showing communication analysis](image)

- **Sendrecv time**
- **Allreduce time**

**X-axis:** Number OPENMP threads per MPI task

**Y-axis:**
- **Time/s**
- **Average Message Size / Bytes**

**Legend:**
- Sendrecv Aver mess
- Allreduce Average Message
Strong Scaling cont.

```!
$omp parallel do private(i,j)
    do i=1,mi
        dnitmp(i,threadid) = ...
$omp critical
    do k=1,nthreads
        do j=1,mgrid
            dni(j) = dni(j)+dni(tmp(j,k))
    .
```
Strong Scaling cont.

```fortran
!$omp parallel do private(i,j)
do i=1,mgrid
  do j=1,nindex(i,k)
    ptilde(i)=ptilde(i)+ring(j,i,k)*phitmp(indexp(j,i,k))
  enddo
enddo
```

- total
- OMP time
- R00025 poisson.f90 (92-100)
- R00015 chargei.F90 (29-74)
- R00053 pushi.f90 (64-111)
- R00054 pushi.f90 (123-236)
- R00016 chargei.F90 (86-161)
PARATEC - First Principles Electronic Structure Calculations

- First Principles: Full quantum mechanical treatment of electrons
- Gives accurate results for Structural and Electronic Properties of Materials, Molecules, Nanostructures
- Computationally very expensive (e.g., grid of > 1 million points for each electron)
- Density Functional Theory (DFT) Plane Wave Based (Fourier) methods probably largest user of Supercomputer cycles in the world.
- ~13% total NERSC workload including single “biggest” code VASP
- PARAllel Total Energy Code (PARATEC) proxy in the NERSC6 benchmark suite
ab initio Density Functional Theory (Kohn 98 Nobel Prize)

Many Body Schrödinger Equation (exponential scaling)

\[ \left\{ -\sum_i \frac{1}{2} \nabla_i^2 + \sum_{i,j} \frac{1}{|r_i - r_j|} + \sum_{i,l} \frac{Z}{|r_i - R_l|} \right\} \Psi(r_1,..r_N) = E \Psi(r_1,..r_N) \]

Kohn Sham Equation (65): The many body ground state problem can be mapped onto a single particle problem with the same electron density and a different effective potential (cubic scaling).

\[ \left\{ -\frac{1}{2} \nabla^2 + \int \frac{\rho(r')}{|r-r'|} dr' + \sum_l \frac{Z}{|r-R_l|} + V_{XC} \right\} \psi_i(r) = E_i \psi_i(r) \]

\[ \rho(r) = \sum_i |\psi_i(r)|^2 = |\Psi(r_1,..r_N)|^2 \]

Use Local Density Approximation (LDA) for \( V_{XC}[\rho(r)] \) (good Si:C)
Load Balancing & Parallel Data Layout

- Wavefunctions stored as spheres of points (100-1000s spheres for 100s atoms)
- Data intensive parts (BLAS) proportional to number of Fourier components
- Pseudopotential calculation, Orthogonalization scales as $N^3$ (atom system)
- FFT part scales as $N^2 \log N$

Data distribution: load balancing constraints (Fourier Space):
- each processor should have same number of Fourier coefficients ($N^3$ calcs.)
- each processor should have complete columns of Fourier coefficients (3d FFT)

$$\frac{1}{2} \nabla^2 \psi_i(r)$$

Give out sets of columns of data to each processor
Basic algorithm & Profile of Paratec

- **Orthogonalization – ZGEMM**
  - $N^3$

- **FFT**
  - $N \, \ln N$

- At small concurrencies ZGEMM dominates at large FFT
What OpenMP can do for Paratec?

- **ZGEMM** very amenable to threading

- **FFT** also
  - Can thread FFT library calls themselves
  - Can ‘package’ individual FFT’s so that messages are combined -> more efficient communication
Parallel “ZGEMM”

Time / s

OpenMP threads / MPI tasks

Computation
Communication

768 384 256 128 64
1 2 3 6 12

41
FFT Breakdown

![Diagram showing FFT Breakdown with OpenMP threads and MPI tasks.](image)
PARATEC - Memory Usage

Memory per node (GB)

OpenMP threads / MPI tasks

<table>
<thead>
<tr>
<th>Threads</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>768 GB</td>
</tr>
<tr>
<td>2</td>
<td>384 GB</td>
</tr>
<tr>
<td>3</td>
<td>256 GB</td>
</tr>
<tr>
<td>6</td>
<td>128 GB</td>
</tr>
<tr>
<td>12</td>
<td>64 GB</td>
</tr>
</tbody>
</table>
• Dynamics and physics use separate decompositions
  – physics utilizes a 2D longitude/latitude decomposition
  – dynamics utilizes multiple decompositions
• FV dynamics 2D block latitude/vertical and 2D block longitude/latitude
• Decompositions are joined with transposes
• Each subdomain is assigned to at most one MPI task
• Additional parallelism via OpenMP ~500 OpenMP directives over 72 .F90 files
fvCAM coordinate system

- 576x361x28 grid (Longitude x Latitude x Vertical) (X Y Z)
- Original problem definition - 240 MPI tasks - 60(Y) x 4(Z,X) decomposition
- Dynamics uses Lat-Vert and Lat-Long
- Physics uses Lat-Long decomposition
fvCAM MPI+OpenMP Performance

- Dynamics
- Physics
- OpenMP
- MPI

Time / s

OpenMP threads / MPI tasks

1  2  3  6  12

240 120 80 40 20

GOOD
fvCAM Physics

<table>
<thead>
<tr>
<th>OpenMP threads / MPI tasks</th>
<th>Time / s</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>240</td>
</tr>
<tr>
<td>2</td>
<td>120</td>
</tr>
<tr>
<td>3</td>
<td>80</td>
</tr>
<tr>
<td>6</td>
<td>40</td>
</tr>
<tr>
<td>12</td>
<td>20</td>
</tr>
</tbody>
</table>
• Columnar processes (typically parameterized) such as precipitation, cloud physics, radiation, turbulent mixing lead to large amounts of work per thread and high efficiency

```c
$OMP PARALLEL DO PRIVATE (C)
do c=begchunk, endchunk
   call tphysbc (ztodt, pblht(1,c), tpert(1,c),
                snowhland(1,c),phys_state(c),phys_tend(c), pbuf,fsds(1,c)....
endo```

fvCAM - Dynamics

OpenMP threads / MPI tasks

Time / s

OpenMP  MPI

1  240
2  120
3  80
6  40
12 20

49
Less Memory Usage with OpenMP Compared to Flat MPI

Memory per Node (GB)

<table>
<thead>
<tr>
<th>OpenMP threads / MPI tasks</th>
<th>Memory per Node (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>240</td>
</tr>
<tr>
<td>2</td>
<td>120</td>
</tr>
<tr>
<td>3</td>
<td>80</td>
</tr>
<tr>
<td>6</td>
<td>40</td>
</tr>
<tr>
<td>12</td>
<td>20</td>
</tr>
</tbody>
</table>

fvCAM
Advanced OpenMP techniques
GTC - Shifte Routine

- Which e\textsuperscript{-} to move?
- Pack e\textsuperscript{-} to be moved
- Communicate \# e\textsuperscript{-} to move
- Repack non-moving e\textsuperscript{-}
- Send/Recv e\textsuperscript{-}
- And again…. 

![Bar Chart](image)

- serial
- openmp
- mpi
Shifte Routine

- Which e\textsuperscript{-} to move? ✓
- Pack e\textsuperscript{-} to be moved ×
- Communicate # e\textsuperscript{-} to move ×
- Repack non-moving e\textsuperscript{-} ×
- Send/Recv e\textsuperscript{-} ×
- And again…..
OPENMP tasking

Executing Thread Encountering Task Region Adds Task to pool
#pragma omp task

Idle Threads Can Execute Tasks in pool
Tasking - Results

Shifter ~30% faster!
GTC overall ~5% faster
Summary

- OpenMP + MPI can be faster than pure MPI – and is often comparable in performance
- Beware NUMA!
  - Don’t use >6 OpenMP threads unless absolutely necessary or you can ‘first-touch’ perfectly
- Beware !$OMP critical!
  - Unless you absolutely have to
- Need Holistic view of your codes performance bottlenecks
  - Adding more cores may not help – transpose
1. Should I use OpenMP?
   + Need to save memory and have duplicated structures across MPI tasks
   + Routine that parallelises with OPENMP only – Poisson routine in GTC
   – Reduction operations – charge & push in GTC
   – Threads can be hard – locks, race conditions

2. How hard is it to change my code?
   • Easier than serial to MPI
   • Easier than UPC/ CAF ?

3. How do I know if it’s working or not?
   – IPM, OMPP, TAU, HPCToolkit, Craypat
Lessons for NERSC Users-
Longer Term

• Are you going to tell me in 3 years that I should have used CAF/UPC/Chapel?
• Uncertainty about Future Machine model
  – GPU programming model – streaming
  – Many lightweight cores
• OpenMP as it stands today is not ideally suited to either model
  – Mend it? Broken ?? (GPU flavor of OMP)
Next Steps for COE

• Phase 1 completing
  – Application studies completed
  – Final technical report by end of year

• Phase 2 starting up
  – Apply lessons from phase 1 to leading application codes
    • VASP: largest user base at NERSC (create OpenMP implementation)
  – Selecting advanced programming models for study
  – Selecting representative applications and kernels from NERSC-6 applications
    • GTS: represents broad class of PIC algorithms (create CAF version)