Best practices for writing and running mix-mode MPI and OpenMP codes on the Cray XE6

LBNL NERSC
Nicholas J Wright, Karl Fuerlinger, John Shalf

LBNL Computing Research Division
Hongzhang Shan, Tony Drummond, Andrew Canning

PPPL
Stephane Ethier

Cray Inc.
Nathan Wichmann, Marcus Wagner, Sarah Anderson, Ryan Olsen, Mike Aamodt
• 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 ↑
  – Memory Capacity per core flat or ↓
  – Memory Bandwidth per FLOP ↓
  – Network Bandwidth per FLOP ↓

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 18\textsuperscript{th} months
  - New Trend: Double # cores every 18 months

- **Implication for NERSC users**
  - 3x increase in system performance with no per-core performance improvement (hopper)
  - 12x more cores in NERSC-6 (hopper) than NERSC-5 (franklin) (2 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 *durable* model that can sustain massive growth in parallelism
  - Hopper changes are first step in a long-term technology trend
  - NERSC needs to take proactive role in guiding transition of user community
Long-Term Concerns for NERSC Users

- Franklin (N5) + QC
  - 19 TF Sustained
  - 36 TF Sustained
  - 352 TF Peak

- COTS/MPP + MPI

- Hopper (N6)
  - >1 PF Peak

- NERSC-7
  - 10 PF Peak

- NERSC-8
  - 100 PF Peak

- NERSC-9
  - 1 EF Peak

- Exascale + ???

- GPU CUDA/OpenCL
  Or Manycore BG/Q, R

- Franklin (N5)
  - 19 TF Sustained
  - 101 TF Peak

- COTS/MPP + MPI (+ OpenMP)

- Top500

- >1 PF Peak
NERSC COE

• Risks for NERSC and DOE User Community
  – Users will not be able to make effective user of hopper
  – Average job size will go down if users cannot scale
  – *Users will be exposed to multiple-disruptive rewrites of their code in effort to stay head of technology curve*

• As mitigation for this risk, NERSC created the Cray Center of Excellence in cooperation with Cray Inc.
  – Characterize performance of NERSC codes in context of emerging technology trends
  – Evaluate viable/candidate programming models to make more effective use of future machines (hopper first)
  – Develop training materials to guide the user transition to new programming model *(map durable path to exascale)*
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 to disseminate results

• **Phase 2: Prepare users for next decade**
  – Evaluate advanced programming models
  – Identify durable approach for programming on path to exascale
<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>x</td>
<td>x IMPACT-T</td>
<td>x IMPACT-T</td>
<td>x IMPACT-T</td>
<td>x</td>
</tr>
<tr>
<td>Astrophysics</td>
<td>x</td>
<td>x MAESTRO</td>
<td>x</td>
<td>x</td>
<td>x MAESTRO</td>
<td>x MAESTRO</td>
</tr>
<tr>
<td>Chemistry</td>
<td>x GAMEESS</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Climate</td>
<td></td>
<td>x CAM</td>
<td></td>
<td></td>
<td>x CAM</td>
<td></td>
</tr>
<tr>
<td>Combustion</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x MAESTRO</td>
<td>x AMR Elliptic</td>
</tr>
<tr>
<td>Fusion</td>
<td>x</td>
<td>x</td>
<td>x GTC</td>
<td>x GTC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lattice Gauge</td>
<td></td>
<td>x MILC</td>
<td>x MILC</td>
<td>x MILC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Material Science</td>
<td></td>
<td>x PARATEC</td>
<td>x PARATEC</td>
<td></td>
<td></td>
<td>x PARATEC</td>
</tr>
</tbody>
</table>
OpenMP Hybrid Programming Basics
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
What are the Basic Differences Between MPI and OpenMP?

• 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.

• 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

K. Yelick, CS267 UCB
Understanding Hybrid MPI/OPENMP Model

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

**Count = G / N_{\text{MPI}}**

Do \( i = 1, \text{count} \)

**Count = G / N_{\text{OMP}}**

!$\text{omp do private (i)}$

Do \( i = 1, G \)

**Count = G / (N_{\text{OMP}} * N_{\text{MPI}})**

!$\text{omp do private (i)}$

Do \( i = 1, G / N_{\text{MPI}} \)

**Count = G**

Do \( i = 1, G \)
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
• Heterogeneous Memory access between dies
• “First touch” assignment of pages to memory.

2xDDR1333 channel
21.328 GB/s

3.2GHz x8 lane HT
6.4 GB/s bidirectional

3.2GHz x16 lane HT
12.8 GB/s bidirectional

• Locality is key *(just as per Exascale Report)*
• Only *indirect* locality control with OpenMP
Hopper Node Topology
Understanding NUMA Effects

- Heterogeneous Memory access between dies
- “First touch” assignment of pages to memory.

2xDDR1333 channel
21.328 GB/s

3.2GHz x8 lane HT
6.4 GB/s bidirectional

3.2GHz x16 lane HT
12.8 GB/s bidirectional

- Locality is key (just as per Exascale Report)
- Only indirect locality control with OpenMP
• Heterogeneous Memory access between dies
• “First touch” assignment of pages to memory.

2xDDR1333 channel
21.328 GB/s

3.2GHz x8 lane HT
6.4 GB/s bidirectional

3.2GHz x16 lane HT
12.8 GB/s bidirectional

• Locality is key (just as per Exascale Report)

Launch threads on “NUMA Nodes” (see COE talk)
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];
}
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

Graph showing bandwidth (GB/s) vs. number of OpenMP threads for 'TouchByAll' and 'TouchByOne' scenarios.
Why does it matter? - NUMA mem latency

Node 0 <-> Node 0...3
lat_mem_rd -P 1 -N 5 18

Latency (ns)

Memory Size (MB)
Studying the N6 Application Benchmarks
NERSC-6 Benchmark Codes

- Gyrokinetic Toroidal Code (GTC)
- Parallel Total Energy Code (PARATEC)
- Finite Volume Community Atmosphere Model (fvCAM)
<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>X</td>
<td>X IMPACT-T</td>
<td>X IMPACT-T</td>
<td>X IMPACT-T</td>
<td>X</td>
</tr>
<tr>
<td>Astrophysics</td>
<td>X</td>
<td></td>
<td>X IMPACT-T</td>
<td>X</td>
<td>X MAESTRO</td>
<td>X MAESTRO</td>
</tr>
<tr>
<td>Chemistry</td>
<td>X GAMESS</td>
<td>x</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Climate</td>
<td></td>
<td></td>
<td>X CAM</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Combustion</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X MAESTRO</td>
<td>X AMR Elliptic</td>
</tr>
<tr>
<td>Fusion</td>
<td>X</td>
<td>x</td>
<td></td>
<td></td>
<td>X GTC</td>
<td>X</td>
</tr>
<tr>
<td>Lattice Gauge</td>
<td>X MILC</td>
<td>X MILC</td>
<td>X MILC</td>
<td></td>
<td>X MILC</td>
<td>X</td>
</tr>
<tr>
<td>Material Science</td>
<td>X PARATEC</td>
<td>X PARATEC</td>
<td>X</td>
<td></td>
<td></td>
<td>X PARATEC</td>
</tr>
</tbody>
</table>

NERSC-6 Applications Cover Algorithm and Science Space
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
Gyrokinetic Toroidal Code (GTC)

- 3D Particle-in-cell (PIC)
- Used for simulations of non-linear gyrokinetic plasma microturbulence
- Paralleled 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) simulations

- Popular method for numerical simulation of many-body systems.
- Often implemented from first principles without the need of an approximate equation of state.
- Applications: plasma modeling, Astrophysics and modeling of debris fields from explosions.
- 1/3 of all CPU hours at NERSC.
• **GTC PIC Steps**
  
  – **Scatter:** deposit charges on the grid (interpolate to nearest neighbor)
  
  – **Solve Poisson equation:** (local relaxation steps)
  
  – **Gather:** forces on each particle from potential
  
  – **Push:** move particles
  
  – 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 – Hopper – Large Test Case

**Graph Details:**
- **Y-axis:** Time (s)
- **X-axis:** OpenMP Threads / MPI tasks
- **Legend:**
  - OpenMP time
  - MPI time
  - Serial time
  - Total Memory / GB

**Data Points:**
- 1 OpenMP thread / 1 MPI task: 1536 s, 2.50E+03 GB
- 2 OpenMP threads / 2 MPI tasks: 768 s, 2.00E+03 GB
- 3 OpenMP threads / 3 MPI tasks: 512 s, 1.50E+03 GB
- 6 OpenMP threads / 6 MPI tasks: 256 s, 1.00E+03 GB
- 12 OpenMP threads / 12 MPI tasks: 128 s, 5.00E+02 GB
- 24 OpenMP threads / 24 MPI tasks: 64 s, 0.00E+00 GB

**Observations:**
- The graph shows a decrease in time as the number of threads increases.
- There is a good balance between time and memory usage.

**Conclusion:**
- The test case demonstrates efficient scaling and memory management with increasing thread counts.
Small Test Case – 96 cores – Breakdown

The diagram shows the breakdown of execution times for different tasks across various numbers of OPENMP threads and MPI tasks. The tasks include:

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

The X-axis represents the combination of OPENMP threads and MPI tasks, while the Y-axis represents the time taken in seconds. The data points indicate how each task behaves under different load conditions.
Small Test Case – 96 cores – Breakdown

The graph shows the time (in seconds) for different processes with varying numbers of OPENMP threads and MPI tasks. The processes include:
- Poisson
- Charge
- Smooth
- Pusher
- Field
- Shift

The x-axis represents the number of OPENMP threads and MPI tasks, ranging from 1 to 12, with corresponding values of 96, 48, 32, 24, 16, and 8. The y-axis represents the time in seconds, ranging from 0 to 200.

The processes are differentiated by color:
- Poisson: Blue
- Charge: Brown
- Smooth: Gold
- Pusher: Gray
- Field: Cyan
- Shift: Orange
Small Test Case – 96 cores – Breakdown
GTC: Communication Analysis

![Graph showing the relationship between the number of OPENMP threads per MPI task and time, as well as the average message size in bytes. The graph includes two plots: one for Sendrecv time and another for Allreduce time. The y-axis represents time in seconds, and the x-axis represents the number of OPENMP threads per MPI task. The graph also shows a separate plot for the average message size in bytes, with two lines indicating Sendrecv Aver mess and Allreduce Average Message.]
Strong Scaling

Time / s

Ncores

pusher
shift
charge
poisson
smooth
total
Strong Scaling cont.
Strong Scaling cont.

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

![Graph showing time vs. number of cores for different files and time components: total, OMP time, and individual files like 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
Many Body Schrodinger Equation (exponential scaling)

\[-\sum \frac{1}{2} \nabla_i^2 + \sum_{i,j} \frac{1}{|r_i - r_j|} + \sum_{i,I} \frac{Z}{|r_i - R_I|} \] \( \Psi(r_1,..r_N) = E \Psi(r_1,..r_N) \)

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).

\[-\frac{1}{2} \nabla^2 + \int \frac{\rho(r')}{|r - r'|} dr' + \sum_{I} \frac{Z}{|r - R_I|} + V_{XC} \] \( \psi_i(r) = E_i \psi_i(r) \)

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

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

- Wavefunctions stored as spheres of points (100-1000 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
Paratec MPI+OpenMP Performance

The graph shows the performance comparison between FFT, "DGEMM", and MPI for different numbers of OpenMP threads and MPI tasks. The x-axis represents the OpenMP threads and MPI tasks combinations, and the y-axis represents the time in seconds. The data points indicate that the performance improves as the number of OpenMP threads and MPI tasks increases, with the best performance achieved at 12 OpenMP threads and 64 MPI tasks.
Parallel “ZGEMM”

- Time / s
  - OpenMP threads / MPI tasks
  - 1: 768
  - 2: 384
  - 3: 256
  - 6: 128
  - 12: 64

- Computation
- Communication
FFT Breakdown

- Computation
- Communication

<table>
<thead>
<tr>
<th>OpenMP threads / MPI tasks</th>
<th>Time / s</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>768</td>
</tr>
<tr>
<td>2</td>
<td>384</td>
</tr>
<tr>
<td>3</td>
<td>256</td>
</tr>
<tr>
<td>6</td>
<td>128</td>
</tr>
<tr>
<td>12</td>
<td>64</td>
</tr>
</tbody>
</table>
Finite Volume Community Atmospheric Model- fvCAM

- 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

Initialization ➔ Transpose ➔ Dynamics1 ➔ Transpose ➔ Dynamics2 ➔ Physics
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 - Hopper

![Graph showing the relationship between OpenMP threads, MPI tasks, time, and memory usage. The graph illustrates a decrease in time and an increase in memory usage as OpenMP threads and MPI tasks increase.]
fvCAM MPI+OpenMP Performance

Time / s

- Dynamics
- Physics
- OpenMP
- MPI

OpenMP threads / MPI tasks

- 1: 240
- 2: 120
- 3: 80
- 6: 40
- 12: 20

G O O D
• 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,fds(1,c)....
enddo
```
fvCAM - Dynamics

OpenMP threads / MPI tasks

57
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)
Advanced OpenMP techniques
• 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…. 
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…..

![Graph showing relative time for serial, openmp, and mpi](image)
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