Experience with MPI/OpenMP on Cray XE6 (hopper)

Sherry Li (LBNL)

FASTMath Webinar, July 11, 2012
Manycore trend

- **Hopper, Cray XE6 @ NERSC**
  - 153,216 total cores, peak 1.28 peta-flops

- **Node architecture**
  - 24 cores (2 x 12-core AMD 'MagnyCours' 2.1 GHz processors)
  - 32 GB memory (~1GB/core user space)
  - Each core has L1 & L2 caches, L3 shared among 12-core
  - 4 DDR3 1333 MHz memory channels per 12-core
Hopper Node Topology

Understanding NUMA Effects [J. Shalf]

- 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
Case 1

Sparse LU in SuperLU_DIST

Ichi Yamazaki, Li, IPDPS 2012 Proceedings
SuperLU_DIST steps to solution:

1. **Matrix preprocessing**
   - static pivoting/scaling/permutation to improve numerical stability and to preserve sparsity

2. **Symbolic factorization**
   - compute e-tree/structure of LU and static comm./comp. scheduling
   - find supernodes (6-80 cols) for efficient dense BLAS operations

3. **Numerical factorization (dominate)**
   - Right-looking, outer-product
   - 2D block-cyclic MPI grid

4. **Triangular solve with forward, back substitutions**
SuperLU_DIST right-looking factorization

for \( j = 1, 2, \ldots, Ns \) (# of supernodes)

\[
// \text{panel factorization (column and row)}
\]
- factor \( A(j,j) \) and isend to \( P_C(j) \) and \( P_R(j) \)
- wait for \( A(j,j) \) and factor column \( A(j+1:Ns, j) \)
  and send down to \( P_R(,:) \)
- wait for \( A(j,j) \) and factor row \( A(j, j+1:Ns) \)
  and send right to \( P_C(,:) \)

\[
// \text{trailing matrix update}
\]
- update \( A(j+1:Ns, j+1:Ns) \)

end for

- Implementation uses flexible look-ahead
- Trailing matrix update has high parallelism, good load-balance, but time-consuming
MPI+OpenMP (fork/join) for trailing update

- each MPI process updates independent supernodal blocks
  - use OpenMP threads to update these blocks

2x2 MPI, 4 threads-w/-task 1 (1D)

2x2 MPI, 4 threads-w/-task 1 (2D)
MPI + OpenMP factorization time on 16 nodes

Accelerator (sym), n=2.7M, fill-ratio=12
Fusion (unsym), n=801K, fill-ratio=10

Best time: 128 x 2

Best time: 128 x 1, or 128 x 2
MPI + OpenMP memory usage on 16 nodes

- LU + user buffer + system

Accelerator (sym), n=2.7M, fill-ratio=12

Fusion (unsym), n=801K, fill-ratio=10
Case 2

- PIC beam dynamics code (3d FFT Poisson solver)
  - Li, Ji Qiang, Rob Ryne, ICIAM 2011 presentation
IMPACT Code Suite (ComPASS-2) [Qiang/Ryne]

- 3D Particle-In-Cell code with two domains: charged particles and the electric field generated by the charged particles
- F90 + MPI, code in development 15+ years

Split-operator method ... Each simulation step:
1) advance particles half-step in **external field** $H_{\text{ext}}$
2) deposit charge density on mesh grid
3) solve Poisson equation for field vector (electric potential)
   - **global communication**
4) interpolate the field vector from grids to particles
5) advance particles in **space-charge field (Coulomb)** $H_{\text{sc}}$
6) advance particles half-step in **external field** $H_{\text{ext}}$
MPI + OpenMP effects

Each simulation step:

1. advance particles half-step in external field $H_{\text{ext}}$
2. deposit charge density on mesh grid
3. solve Poisson equation for field vector (electric potential)
4. interpolate the field vector from grids to particles
5. advance particles in space-charge field (Coulomb) $H_{\text{sc}}$
6. Advance particles half-step in external field $H_{\text{ext}}$
Compute external field $H_{\text{ext}}$

- Stored in a table: $\text{Table}(9, \text{Nparticles})$
- Entirely parallel, no need for synchronization

```c
!$OMP PARALLEL DO private (i, tmppx, tmppy, . . . )
!$OMP default(shared)
   do i = 1, Nparticles
      tmppx = . . .
      tmppy = . . .
      . . .
      $\text{Table}(1, i) = . . .$
      $\text{Table}(2, i) = . . .$
      . . .
   enddo
!$OMP END PARALLEL DO
```
Time to compute $H_{\text{ext}}$

- Hopper @ NERSC, 24 cores per node
- 1 billion particles, 129 x 129 x 2048 mesh, 1 turns in a ring
- Experiment: use 128 nodes, trade thread with MPI process

![Computing linear map](chart.png)

- Total memory usage (GB) reduced:
  
  $410 \rightarrow 313 \rightarrow 241 \rightarrow 214 \rightarrow 201$
MPI + OpenMP effects

Each simulation step:

1. advance particles half-step in external field $H_{\text{ext}}$
2. deposit charge density on mesh grid
   - solve Poisson equation for field vector (electric potential)
4. interpolate the field vector
5. advance particles in space-charge field (Coulomb) $H_{\text{sc}}$
6. Advance particles half-step in external field $H_{\text{ext}}$
Poisson solver using 3d FFT

Steps of parallel 3D FFT:

- multiple 1D FFTs along X (local)
- Transpose (communication)
- multiple 1D FFTs along Y (local)
- Transpose (communication)
- multiple 1D FFTs along Z (local)
- (optional) Transpose (communication)
Transpose

1) Copy 3d array to 1d send buffer.
   !$omp parallel do private(I, …)
   do i = 0, Ntasks – 1
     do m = . . .
       do j = . . .
         do k = . . .
           sendbuf( i + calc_index(m, j, k) ) = x(k, j, m)
         enddo
       enddo
     enddo
   enddo
$omp end parallel do

2) MPI_Alltoallv(sendbuf, recvbuf, . . .)

3) Retrieve from 1d receive buffer to 3d array
   !$omp parallel do private(I, …)
       . . .
   !$omp end parallel do
Transpose time

- hopper @ NERSC, 24 cores per node
- 1 billion particles, 129 x 129 x 2048 mesh
- Experiment: use 128 nodes, trade thread with MPI process

**Benefit to MPI_Alltoallv:**
- Involves fewer messages, larger size per message
- Alltoallv time reduced from 70% to 40% of the transpose time
Remarks

Trading MPI process with OpenMP thread
- More beneficial to memory footprint reduction
- Can be faster than pure MPI, or comparable
- Intra-node: no control of memory & core affinity, runtime maybe severely affected by NUMA
  - Hopper: 3-6 threads better than >6 threads

Need more flexible hybrid programming model than simple MPI-OpenMP-trading
- Dynamic growth & shrink of MPI processes
- Bi-modal MPI