Scaling Parallel 3-D FFT with Non-Blocking MPI Collectives

Talk at ScalA’14

Sukhyun Song and Jeff Hollingsworth

Department of Computer Science
University of Maryland
Fast Fourier Transform (FFT)

• Converts a Function: Time $\rightarrow$ Frequency
  – To solve difficult mathematical problems

• 3-D FFT in Computational Science

Astrophysical N-Body Simulations

Turbulent Flow Simulations
Parallel 3-D FFT

• Works on a 3-D Array of Complex Numbers
• Methods of Parallelization
  – We use 2-D decomposition to increase scaling.

1-D Decomposition

2-D Decomposition
Computation-Communication Overlap

• Expensive Computation and Communication
  – Convert all the complex numbers in a 3-D array
  – Perform all-to-all exchanges

• Hide Communication behind Computation

![Diagram showing computation and communication over time]

- Computation1
- Blocking Communication
- Computation2
- Non-Blocking Communication

Performance increase
Our Approach

• 2-D Domain Decomposition
  – Increase scaling to a large number of cores

• Non-Blocking MPI All-to-All Operation (MPI-3)
  – Exploit computation-communication overlap
  – Rely on optimized communication of MPI collectives

• Characteristics
  – Design an optimized parallel 3-D FFT algorithm
    • Computation-communication overlap
    • Cache reuse
  – Ensure message progress for non-blocking communication
    • No support from special hardware
  – Define and auto-tune parameters
    • Optimize our code in different system environments

• Faster than Other Approaches
  – FFTW (1.83X)
Existing Approach with No Overlap

- Decompose a 3-D Array along Two Dimensions
- Three Steps of Local 1-D FFT
- Two Steps of Blocking All-to-All Communication
  - Multiple process groups for communication

![Diagram of 3D array and FFT operations](image)
Our Approach with Overlap

• Divide a Sub-Array Again into Small Tiles
• Use Non-Blocking All-to-All Communication
• Repeat Comp-Comm-Comp over Tiles
  – Distribute FFTy into two phases
  – Tunable parameters T1 and T2 (tile sizes)
No Tiling, No Overlap

FFTz  A2A  FFTy1

8
Tiling, No Overlap

tile (i-2)
tile (i-1)
tile (i)
tile (i+1)
tile (i+2)
Tiling, Overlap
Computation-Communication Overlap

• Non-Blocking A2A
  – Between `MPI_Ialltoall` and `MPI_Wait`

• Do Computations on Other Tiles during A2A
  – Parameter W1: communication window
Fully Asynchronous Communication

• **MPI_Ialltoall()** Performs Multiple Rounds of Point-to-Point Communication.
  – Need to make them forward progress

• Manually Call **MPI_Test()** → High Portability!
  – Parameters for frequency
Loop Tiling

- Divide a Tile Again into Sub-Tiles
  - Fit a sub-tile in a cache
  - Parameters for sub-tile sizes

- Optimize Cache Performance
  - Pack and unpack for A2A
Auto-Tuning Parameters

• 24 Parameters
  – Two communication tile sizes
  – Two communication window sizes
  – Eight `MPI_Test()` frequencies
  – Eight sub-tile sizes
  – …

• Why Auto-Tuning?
  – 10X performance variance
  – A huge # possible configurations
  – Various system environments
Nelder-Mead Simplex Method
- Build a simplex (hyper-triangle) in a search space
- Measure the performance of simplex points
- Update a simplex by replacing the worst point
  - Until the simplex converges to a single point
## Related Work

<table>
<thead>
<tr>
<th>method</th>
<th>MPI / UPC</th>
<th>1-D / 2-D</th>
<th>overlap</th>
<th>overlap portability</th>
<th>auto-tuning</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFTW (IEEE’05)</td>
<td>MPI</td>
<td>1-D</td>
<td>X</td>
<td></td>
<td>O</td>
</tr>
<tr>
<td>Hoefler et. al (SPAA’08)</td>
<td>MPI</td>
<td>1-D</td>
<td>O</td>
<td>O</td>
<td>X</td>
</tr>
<tr>
<td>Song et. al (PPoPP’14)</td>
<td>MPI</td>
<td>1-D</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>P3DFFT (SIAM JoSC’12)</td>
<td>MPI</td>
<td>2-D</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>2DECOMP&amp;FFT (CUG’10)</td>
<td>MPI</td>
<td>2-D</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Nishtala et. al (IPDPS’09)</td>
<td>UPC</td>
<td>2-D</td>
<td>O</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Song et. al (ScalA’14)</td>
<td>MPI</td>
<td>2-D</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>Kandalla et. al (JoCS R&amp;D’11)</td>
<td>MPI</td>
<td>2-D</td>
<td>inter-array</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
Experimental Setup

• Platform
  – Edison
    • 133,824-core Cray XC30 machine at NERSC
    • Two 12-core Intel “Ivy Bridge” 2.4 GHz processors / node
    • Cray Aries with Dragonfly topology
    • Cray Message Passing Toolkit 6.2.0

• Parallel 3-D FFT Methods
  – NEW: 2-D decomposition, overlap (Active Harmony)
  – FFTW: 1-D decomposition, no overlap (built-in tuner)
  – DCMP: 2-D decomposition, no overlap (process grid)
  – UPCF: 2-D decomposition, overlap, UPC (process grid)
Speedup of NEW over Other Methods

1.83x over FFTW

1.58x over DCMP

1.32x over UPCF

3-D FFT speedup of NEW relative to other algorithms (times)

the number of cores $p$ and input size $N^3$
Strong Scaling

- $N^3 = 1024^3$, $p = 128 - 32768$

NEW scales well like DCMP and UPCF. FFTW finishes scaling here. NEW is the fastest.
Weak Scaling

- $N^3 = 512^3 - 4096 \times 4096 \times 2048$, $p = 128 - 32768$
Conclusions

• Parallel 3-D FFT
  – Increase scaling
    • 2-D domain decomposition
  – Use the non-blocking MPI all-to-all operation
    • Exploit computation-communication overlap
  – Define and auto-tune parameters
  – Performs faster than other approaches

• Future Work
  – Other auto-tuning techniques
  – Release code
Scaling Parallel 3-D FFT with Non-Blocking MPI Collectives

Talk at ScalA’14

Sukhyun Song and Jeff Hollingsworth

Department of Computer Science
University of Maryland
Backup Slides
Overlap Effect

reduced MPI_Wait() time
Auto-Tuning Cost

• Negligible in scientific simulations (>1M timesteps)

39 steps by losing only 5% performance

102 steps for the best configuration