AccFFT: A New Parallel FFT Library for CPU/GPU Architectures

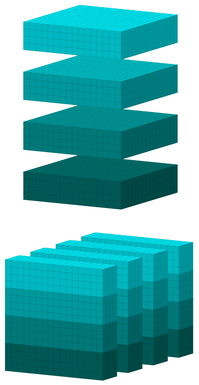
Amir Gholami, George Biros (Advisor)
The University of Texas at Austin

Contributions

- A new massively parallel, distributed FFT library for CPU/GPU that supports multidimensional decomposition
- In house, auto-tuned, k-way all-to-all for MPI communication
- Extension of the k-way all-to-all algorithm to GPUs
- Effective overlap of GPU k-way all-to-all, send/recv operations with PCIe transfers
- Extension of pencil decomposition to high dimensional decomposition for nD FFTs
- Consistent speedup of $2-3\times$, compared to P3DFFT and PFFT libraries
- Consistent speedup of $2\times$ compared to FFTE library (GPU only)
- Source code released under GNU GPL 2 at http://accfft.org

Slab Decomposition

- Decompose the problem along the first dimension
- Local 2D FFT computation followed by a global transpose
- Local 1D FFT computation along the last dimension
- Scalability restricted to $p = N_0$
  - In the limit, each process gets a single 2D slab of the data.



Algorithm 1: Forward slab decomposition algorithm

Input : Data in spatial domain.
Layout: $N_0/P \times N_1 \times N_2$

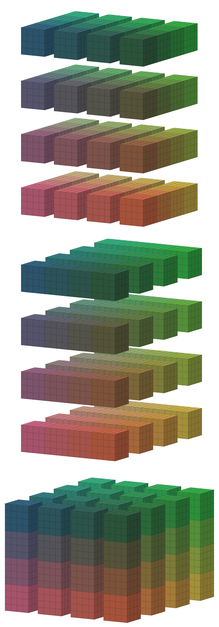
Output: Data in frequency domain.
Layout: $\widehat{N}_0 \times \widehat{N}_1/P \times \widehat{N}_2$

$N_0/P \times \widehat{N}_1 \times \widehat{N}_2 \xleftarrow{FFT} N_0/P \times N_1 \times N_2$;
$N_0 \times \widehat{N}_1/P \times \widehat{N}_2 \xleftarrow{T} N_0/P \times \widehat{N}_1 \times \widehat{N}_2$;
$\widehat{N}_0 \times \widehat{N}_1/P \times \widehat{N}_2 \xleftarrow{FFT} N_0 \times \widehat{N}_1/P \times \widehat{N}_2$;

Pencil Decomposition

- Decompose the problem along the first two dimensions
- Each process gets a batch of pencils, local in one dimension
- Global FFT is computed by three Local 1D FFT computations followed by two global transposes
- Time is dominated by communication overhead for global transposes
- Scalability extended to $p = N_0 \times N_1$
  - In the limit, each process gets a single 1D pencil of the data.

The algorithm can be extended in a similar fashion for computing FFT of nD dimensional arrays.



Algorithm 2: Forward FFT algorithm for pencil decomposition

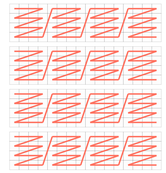
Input : Data in spatial domain.
Layout: $N_0/P_0 \times N_1/P_1 \times N_2$

Output: Data in frequency domain.
Layout: $\widehat{N}_0 \times \widehat{N}_1/P_0 \times \widehat{N}_2/P_1$

$N_0/P_0 \times N_1/P_1 \times \widehat{N}_2 \xleftarrow{FFT} N_0/P_0 \times N_1/P_1 \times N_2$;
$N_0/P_0 \times N_1 \times \widehat{N}_2/P_1 \xleftarrow{T} N_0/P_0 \times N_1/P_1 \times \widehat{N}_2$;
$N_0/P_0 \times \widehat{N}_1 \times \widehat{N}_2/P_1 \xleftarrow{FFT} N_0/P_0 \times N_1 \times \widehat{N}_2/P_1$;
$N_0 \times \widehat{N}_1/P_0 \times \widehat{N}_2/P_1 \xleftarrow{T} N_0/P_0 \times \widehat{N}_1 \times \widehat{N}_2/P_1$;
$\widehat{N}_0 \times \widehat{N}_1/P_0 \times \widehat{N}_2/P_1 \xleftarrow{FFT} N_0 \times \widehat{N}_1/P_0 \times \widehat{N}_2/P_1$;

Optimizing Communication

The communication phase is the most expensive phase of distributed FFT. The pencil decomposition method involves two global transpose operations, which the scaling bottleneck for large processor counts. We use novel communication algorithms, along with efficient (un)packing schemes, to overcome this barrier.



K-way All-To-All

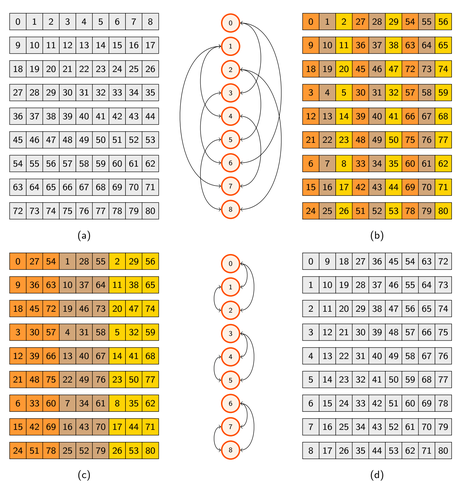
We have optimized our k-way all-to-all and extended it to GPUs. For the GPU version we interleave MPI send/recv with CPU/GPU transfers to effectively hide PCI-e overhead.

Hypercube all-to-all

- Each task sends/receives data from one other task and splits the problem into two smaller ones.
- Stages: $\mathcal{O}(\log p)$
- Total data transfer: $\mathcal{O}(N \log p)$

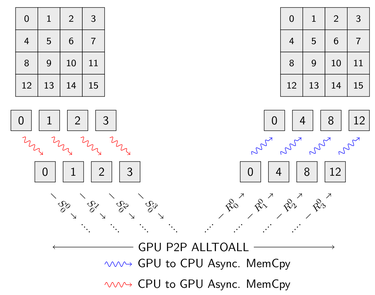
K-way all-to-all

- Each task sends/receives data to/from k-tasks and splits the problem into k smaller ones.
- Stages: $\mathcal{O}(\log p/\log k)$
- Total data transfer: $\mathcal{O}(N \log p/\log k)$
- Overall complexity: $T_{kway} = (t_l k + t_b N/p)\frac{\log p}{\log k}$



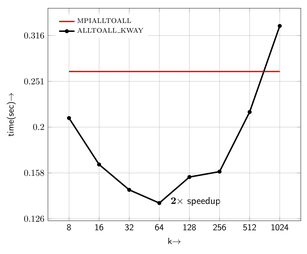
Extension to GPUs

Extension of k-way all-to-all to GPUs by overlapping send/recvs with GPU/CPU data transfers. This effectively hides PCI-e overhead.

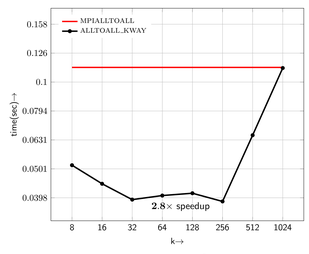


K-way Performance

The optimal splitting parameter, k, can be obtained during setup time. The optimal value depends on system architecture, transpose size, and the number of processes.

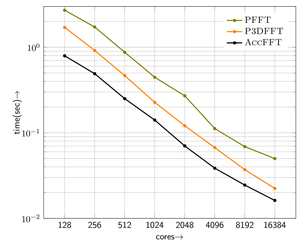


Transpose time for $N = 2048^3$ on 16,384 cores of Stampede

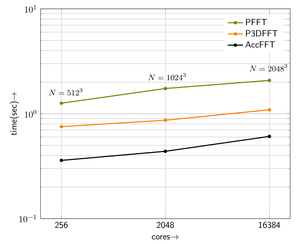


Transpose time for $N = 2048^3$ on 512 GPUs of Titan.

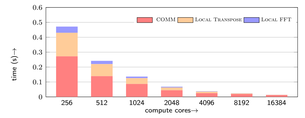
Scaling on Stampede



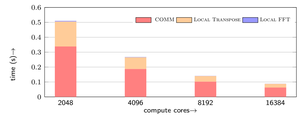
(a) strong scaling for $N = 1024^3$



(c) weak scaling



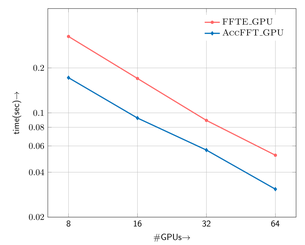
(b) time break down for $N = 1024^3$



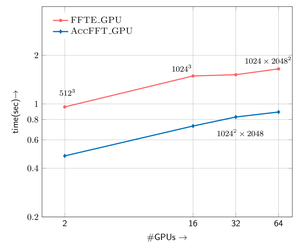
(d) time break down for $N = 2048^3$

CPU scaling results of AccFFT on Stampede for computing a forward R2C FFT.

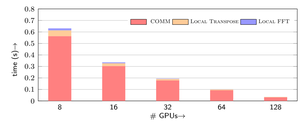
Scaling on Maverick



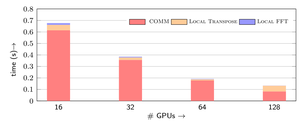
(a) C2C strong scaling for $N = 256 \times 512 \times 1024$



(c) C2C weak scaling



(b) GPU R2C time break down for $N = 1024^3$



(d) GPU C2C time break down for $N = 1024^3$

Scaling results of AccFFT performed on Maverick.

Scaling on Titan

Timing of CPU R2C forward FFT for $N = 2048^3$.

# Cores	Total	Comm	FFT	(Un)Pack
4096	1.14	0.846	0.151	0.142
8192	0.898	0.735	0.0798	0.077
16K	0.375	0.297	0.0377	0.037
32K	0.368	0.297	0.0376	0.014
65K	0.232	0.198	0.0202	0.009
131K	0.116	0.0974	0.0115	0.004

Timing of GPU R2C forward FFT for $N = 1024^3$.

# GPUs	Total	Comm	FFT	(Un)Pack
128	0.138	0.131	0.00097	0.0052
256	0.078	0.073	0.00054	0.0046
512	0.050	0.046	0.00027	0.0025
1024	0.032	0.028	0.00020	0.0025
2048	0.027	0.025	0.00020	0.0014
4096	0.016	0.014	0.00016	0.0018

References

[1] H. Sundar, D. Malhotra, and G. Biros, *HykSort: A new variant of hypercube quicksort on distributed memory architectures*. ICS 13, (2013).

[2] Amir Gholami, Judith Hill, Dhairya Malhotra, and George Biros. *AccFFT: A library for distributed-memory 3-D FFT on CPU and GPU architectures*. (submitted).