Performance Optimization of Multithreaded 2D FFT on Multicore Processors

Challenges and Solution Approaches

Ravi Reddy Manumachu
Research Fellow
(ravi.manumachu@ucd.ie)

School of Computer Science
University College Dublin, Ireland

Overview

- Challenges illustrated using experiments on a modern multicore processor
- Solution methods
  - Parallel computing using load balancing
  - Parallel computing using load imbalancing
- Conclusions and Future work

Experimental Platform

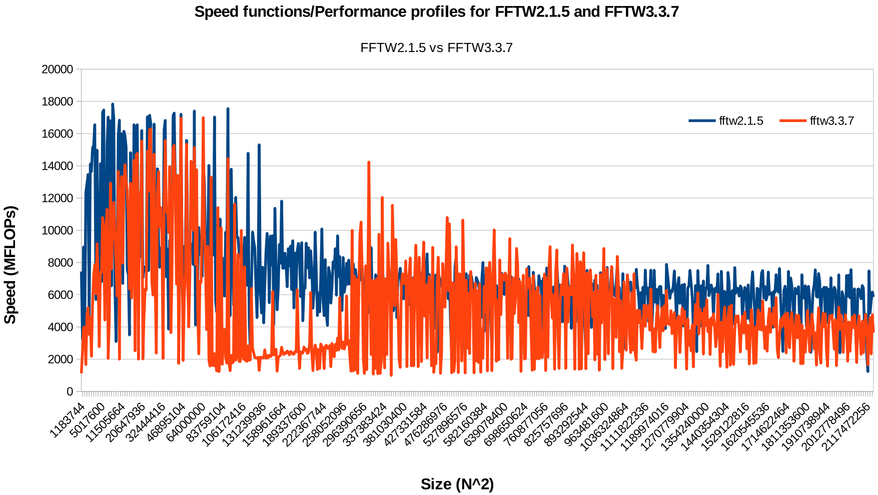
Technical Specifications	Intel Haswell Multicore Processor
Processor	Intel Xeon CPU E5-2699 v3 @ 2.30GHz
OS	CentOS 7.1.1503
Microarchitecture	Haswell
Memory	256 GB
Core(s) per socket	18
Socket(s)	2
NUMA node(s)	2
L1d cache	32 KB
L1i cache	32 KB
L2 cache	256 KB
L3 cache	46080 KB
NUMA node0 CPU(s)	0-17,36-53
NUMA node1 CPU(s)	18-35,54-71

Table: Specification of the Intel Haswell Multicore Profile used to construct the performance profiles.

Overview of applications

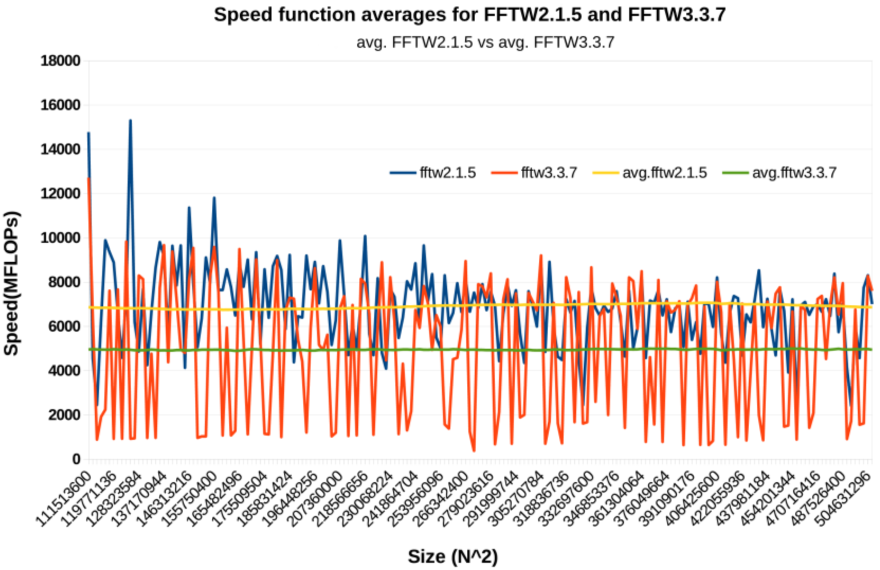
- FFTW-2.1.5 2D-FFT application.
  - Executed using 72 threads.
  - FFT flags (FFTW_ESTIMATE).
- FFTW-3.3.7 2D-FFT application.
  - Executed using 72 threads.
  - FFT flags (FFTW_ESTIMATE).
- Intel MKL 2D-FFT application.
  - Executed using 36 threads.
  - FFT flags (FFTW_ESTIMATE).
- All three applications compute 2D-DFT of a complex signal matrix of size $N \times N$.
- Each thread is bound to a core using *numactl*.
- Performance $= 5.0 \times N^2 \times log_2(N^2)$.
- Test dataset contains 1000 problem sizes ranging from $N = 128$ to $N = 64000$.

Challenges - FFTW-2.1.5 vs FFTW-3.3.7



Performance profiles of FFTW-2.1.5 and FFTW-3.3.7 computing 2D-DFT of size $N \times N$.

Challenges - FFTW-2.1.5 vs FFTW-3.3.7 - Averages

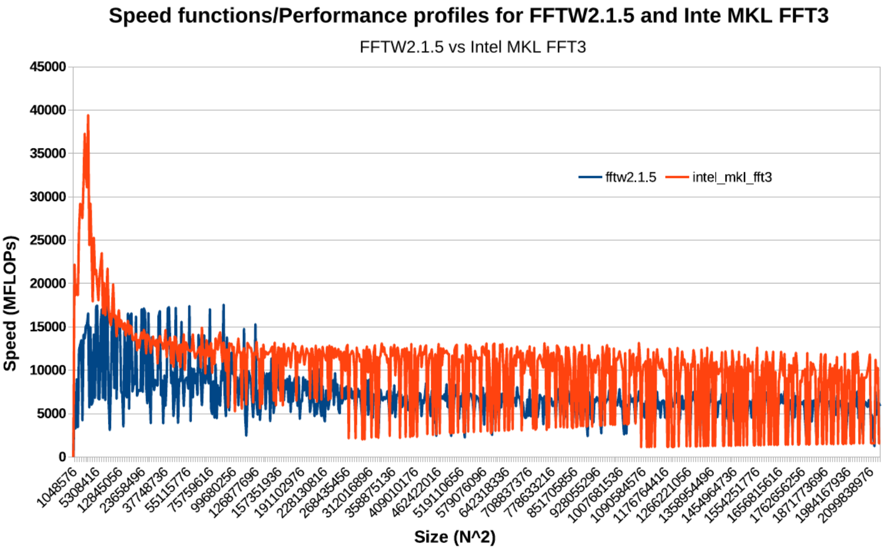


The average performances of FFTW-2.1.5 vs FFTW-3.3.7.

Challenges - FFTW-2.1.5 vs FFTW-3.3.7 - Averages

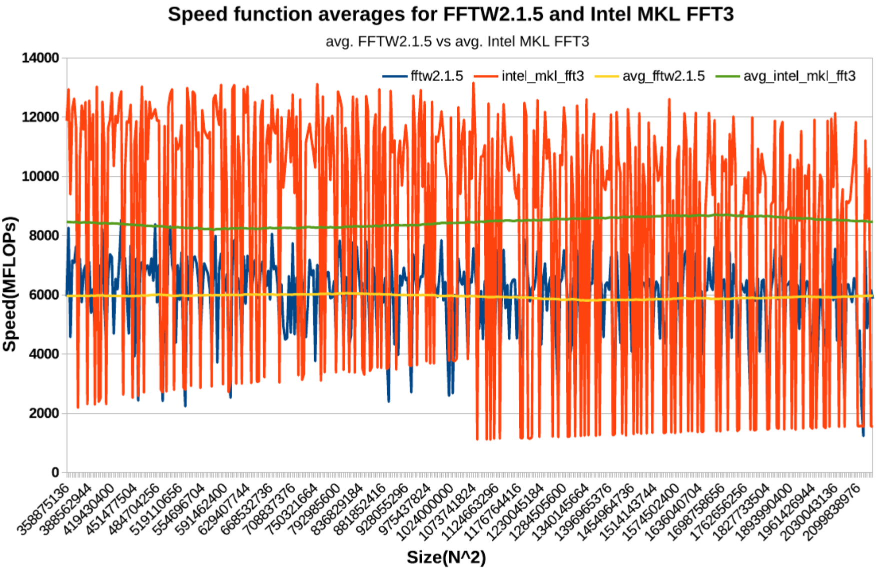
- Peak performances of FFTW-2.1.5 and FFTW-3.3.7 are (17841,16989) MFLOPs.
- Average performances of FFTW-2.1.5 and FFTW-3.3.7 are (7033,5065) MFLOPs.
- FFTW-2.1.5 is better than FFTW-3.3.7 for over 529 problem sizes (out of 1000).

Challenges - FFTW-2.1.5 vs Intel MKL FFT



Performance profiles of FFTW-2.1.5 and Intel MKL FFT computing 2D-DFT of

Challenges - FFTW-2.1.5 vs Intel MKL FFT - Averages

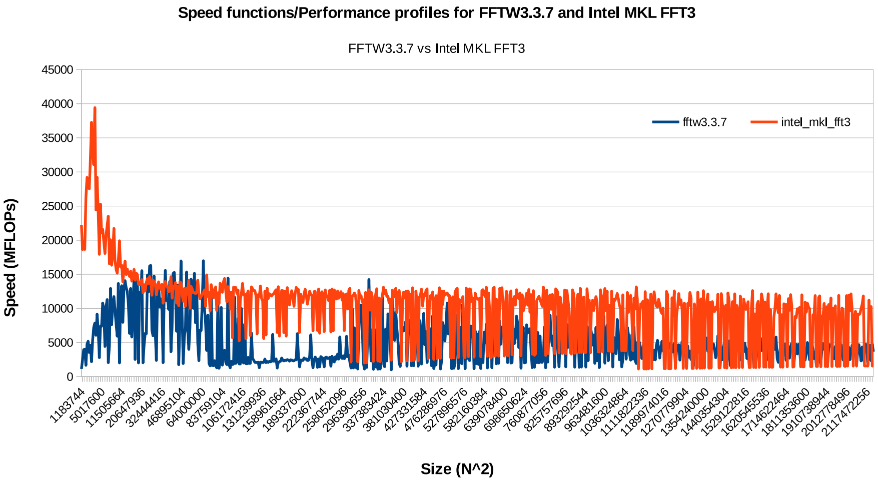


The average performances of FFTW-2.1.5 vs FFTW-3.3.7

Challenges - FFTW-2.1.5 vs Intel MKL FFT - Averages

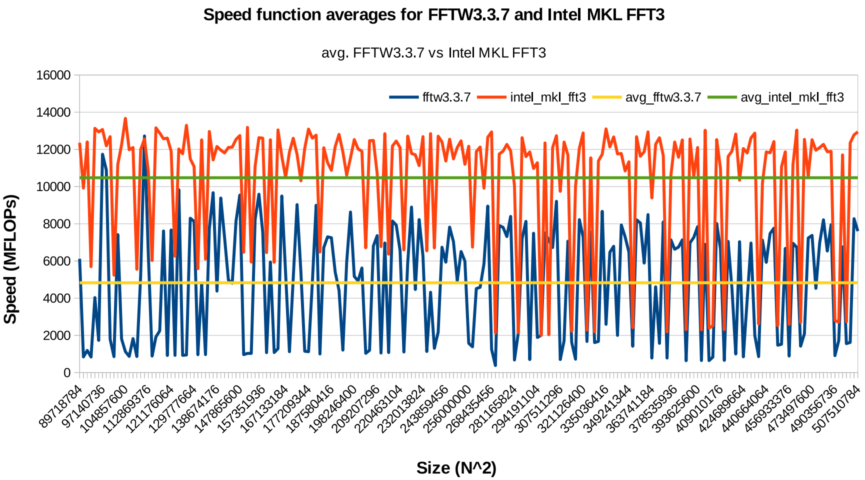
- Peak performances of FFTW-2.1.5 and Intel MKL FFT are (17841,39424) MFLOPs.
- Average performances of FFTW-2.1.5 and Intel MKL FFT are (7033,9572) MFLOPs.
- FFTW-2.1.5 is better than Intel MKL FFT for over 162 problem sizes (out of 1000).

Challenges - FFTW-3.3.7 vs Intel MKL FFT



Performance profiles of FFTW-2.1.5 and Intel MKL FFT computing 2D-DFT of size $N \times N$.

Challenges - FFTW-3.3.7 vs Intel MKL FFT - Averages



The average performances of FFTW-2.1.5 vs FFTW-3.3.7.

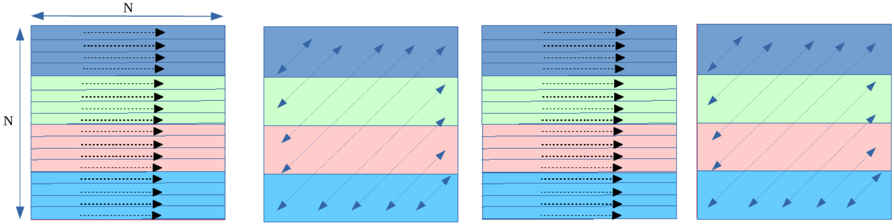
Challenges - FFTW-3.3.7 vs Intel MKL FFT - Averages

- Peak performances of FFTW-3.3.7 and Intel MKL FFT are (16989,39424) MFLOPs.
- Average performances of FFTW-3.3.7 and Intel MKL FFT are (5065,9572) MFLOPs.
- FFTW-3.3.7 is better than Intel MKL FFT for over 199 problem sizes (out of 1000).

Solution approaches

- Optimization through source code analysis and tuning.
- Optimization using solutions for larger problem sizes with better performance.
- Optimization using model-based parallel computing (*will be covered here*).

Parallel computing using load balancing (PFFT-LB)



(a) (b) (c) (d)

PFFT-LB computing 2D-DFT of signal matrix $\mathcal{M}$ of size $N \times N$ $(N = 16)$ using four identical processors. Each processor gets four rows each.

Parallel computing using load balancing (PFFT-LB)

- Parallel algorithm is based on the sequential algorithm employing *row decomposition method*.
- It is executed using p processors. It consists of four steps.
- **Step 1.** Processor P_i executes sequential 1D-FFTs on rows $(i-1) \times \frac{N}{p} + 1, ..., i \times \frac{N}{p}$.
- **Step 2.** Transpose the matrix $\mathcal{M}$.
- **Step 3.** Processor P_i executes sequential 1D-FFTs on rows $(i-1) \times \frac{N}{p} + 1, ..., i \times \frac{N}{p}$.
- **Step 4.** Transpose the matrix $\mathcal{M}$.

PFFT-LB using FFTW-3.3.7 on Intel Haswell Server

- PFFT-LB is executed using p groups of t threads each. A group of t threads constitutes a processor.
- All combinations where $(p \times t = 36)$ and $(p \times t = 72)$ are considered.
- Combinations (p, t) for $p \times t = 36$
  - $(1, 36), (2, 18), ...(36, 1)$
- Combinations (p, t) for $p \times t = 72$
  - $(1, 72), (2, 36), ...(72, 1)$

PFFT-LB: pseudocode for Steps 1,3 using FFTW-3.3.7

```c
    fftw_init_threads();
    fftw_plan_with_nthreads(t);
    fftw_plan plan1 = fftw_plan_many_dft(..., FFTW_ESTIMATE);
    fftw_plan plan2 = fftw_plan_many_dft(..., FFTW_ESTIMATE);
    ...
    fftw_plan planp = fftw_plan_many_dft(..., FFTW_ESTIMATE);

#pragma omp parallel sections num_threads(p)
{
    #pragma omp section
    {
       fftw_execute(plan1);
       fftw_destroy_plan(plan1);
    }
    #pragma omp section
    {
       fftw_execute(plan2);
       fftw_destroy_plan(plan2);
    }
    ...
    #pragma omp section
    {
       fftw_execute(planp);
       fftw_destroy_plan(planp);
    }
}
    fftw_cleanup_threads();
```

PFFT-LB: pseudocode for Steps 1,3 using Intel MKL FFT

```
    fftw_init_threads();
    fftw_plan_with_nthreads(t);

#pragma omp parallel sections num_threads(p)
{
    #pragma omp section
    {
       fftw_plan plan1 =  fftw_plan_many_dft(..., FFTW_ESTIMATE);
       fftw_execute(plan1);
       fftw_destroy_plan(plan1);
    }
    #pragma omp section
    {
       fftw_plan plan1 =  fftw_plan_many_dft(..., FFTW_ESTIMATE);
       fftw_execute(plan2);
       fftw_destroy_plan(plan2);
    }
    ...
    #pragma omp section
    {
       fftw_plan plan1 =  fftw_plan_many_dft(..., FFTW_ESTIMATE);
       fftw_execute(planp);
       fftw_destroy_plan(planp);
    }
}
    fftw_cleanup_threads();
```

PFFT-LB: Performance improvements

- Average performance for FFTW-3.3.7 is 7041 MFLOPs. Best combination varies with problem size.
- Average speedup for FFTW-3.3.7 is 2.7x. Maximum speedup is 6.8x.
- Average performance for Intel MKL FFT is 10818 MFLOPs. Best combination is 2 teams of 18 threads each.
- Average speedup for Intel MKL FFT is 1.4x. Maximum speedup is 2x.

Parallel computing using load imbalancing (PFFT-FPM-LIMB)

- PFFT-FPM-LIMB is executed using p identical processors.
- Inputs are:
  - Signal matrix of size $N \times N$.
  - Speed functions, $S = \{S_1, \cdots, S_p\}$, where $S_i = \{s_i(x_1, y_1), \cdots, s_i(x_m, y_m)\}$ is the speed function of processor P_i.
- $s_i(x, y)$ represents the speed of execution of x number of 1D-FFTs of length y by the processor P_i. $s_i(x, y) = \frac{5.0 \times x \times y \times log_2(y)}{t}$.
- t is the time of execution of x number of 1D-FFTs of length y.
- Output is the transformed signal matrix.

PFFT-FPM-LIMB: Main steps

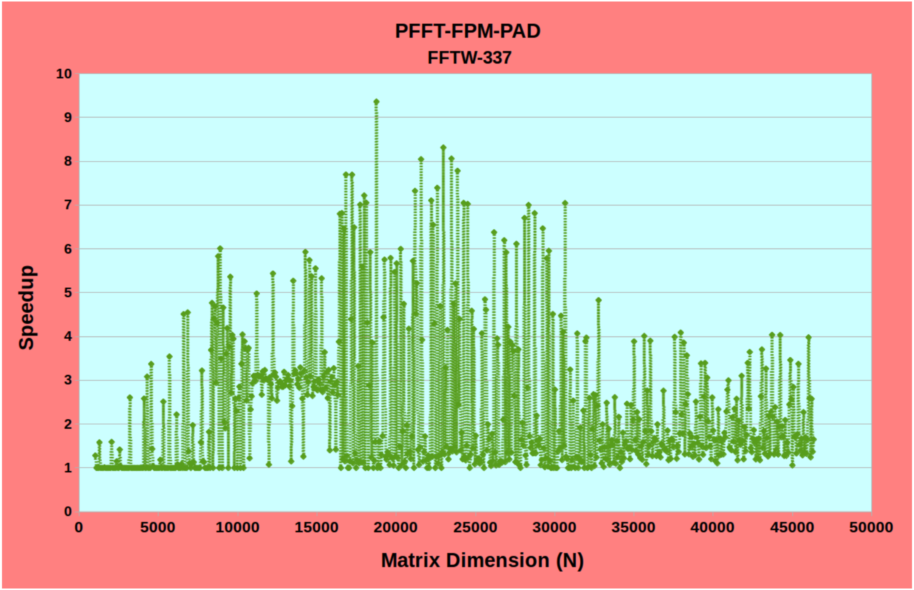
Main steps to execute 2D FFT of size $N \times N$.

- **Step 1.** Partition rows.
  - Speed functions S are sectioned by the plane $y = N$.
  - A set of p curves on this plane is produced which represents the speed functions against variable x given parameter $y = N$ is fixed.
  - Now the number of rows N is unevenly distributed between the processors using the p speed curves as input.
  - The workload partition of N is returned in d where $d[i]$ contains the number of rows owned by Processor P_i.
  - Each row in $d[i]$ is padded by a length l_{pad} where $\left(\frac{d[i] \times l_{pad}}{s_i(d[i], l_{pad})} < \frac{d[i] \times N}{s_i(d[i], N)}\right)$.

PFFT-FPM-LIMB: Main steps... continued...

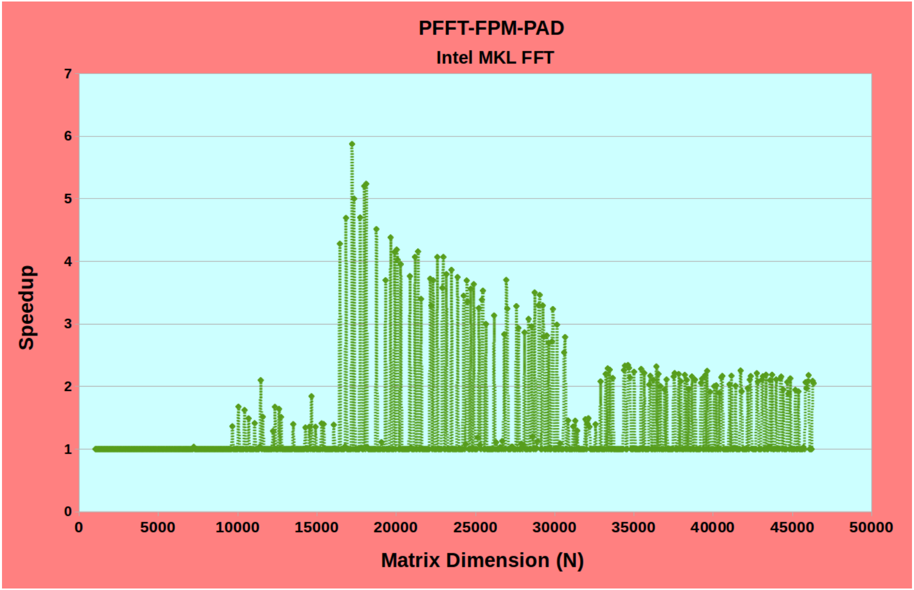
- **Step 2.** Processor P_i executes sequential 1D-FFTs on its padded rows $\sum_{k=1}^{i-1} d[i] + 1, \cdots, \sum_{k=1}^{i} d[i]$.
- **Step 3.** The signal matrix M (excluding the padded region) is transposed.
- **Step 4.** Processor P_i executes sequential 1D-FFTs on its padded rows $\sum_{k=1}^{i-1} d[i] + 1, \cdots, \sum_{k=1}^{i} d[i]$.
- **Step 5.** Same as Step 3.

PFFT-FPM-LIMB: Speedup for FFTW-3.3.7



Speedup of PFFT-FPM-LIMB for FFTW-3.3.7

PFFT-FPM-LIMB: Speedup for Intel MKL FFT



Speedup of PFFT-FPM-LIMB for Intel MKL FFT

PFFT-FPM-LIMB: Summary

- Average performance for FFTW-3.3.7 is 7297 MFLOPs.
- Average speedup for FFTW-3.3.7 is 3x. Maximum speedup is 9.4x.
- Average performance for Intel MKL FFT is 11170 MFLOPs.
- Average speedup for Intel MKL FFT is 2.7x. Maximum speedup is 5.9x.
- Intel MKL FFT is on an average 55% better than FFTW-3.3.7.
- There are 81 problem sizes (out of 1000) where FFTW-3.3.7 is better than Intel MKL FFT.
- Improvement of average performance of FFTW-3.3.7 by 42% over FFTW-2.1.5.
- Improvement of average performance of Intel MKL FFT by 24% over FFTW-2.1.5.

Conclusions and Future work

- Software implementations are available at: https://git.ucd.ie/manumachu/hcllimb.
- For large problem sizes, major variations in performance still remain for FFTW-3.3.7 and Intel MKL FFT. We are exploring solutions to remove them.
- Optimization of 3D FFT using the same methods.
- Understanding the better performance demonstrated by some teams using performance monitoring counts or other debugging tools.