Optimizing the Fast Fourier Transform on a Multi-core Architecture

Long Chen, Ziang Hu, Junmin Lin, Guang R. Gao

IEEE International Parallel and Distributed Processing Symposium, 2007.

Presentation by: Yuanyuan Ding

Dept of Computer & Information Sciences

University of Delaware

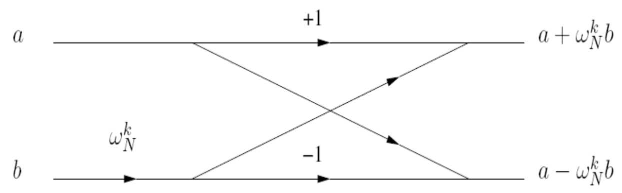
CISC 879 : Software Support for Multicore Architectures

Outline

- FFT Introduction
- IBM Cyclops-64 (C64) Architecture
- 1D FFT Optimization: Step-by-Step
- 2D FFT Optimization
- Conclusion

FFT Introduction

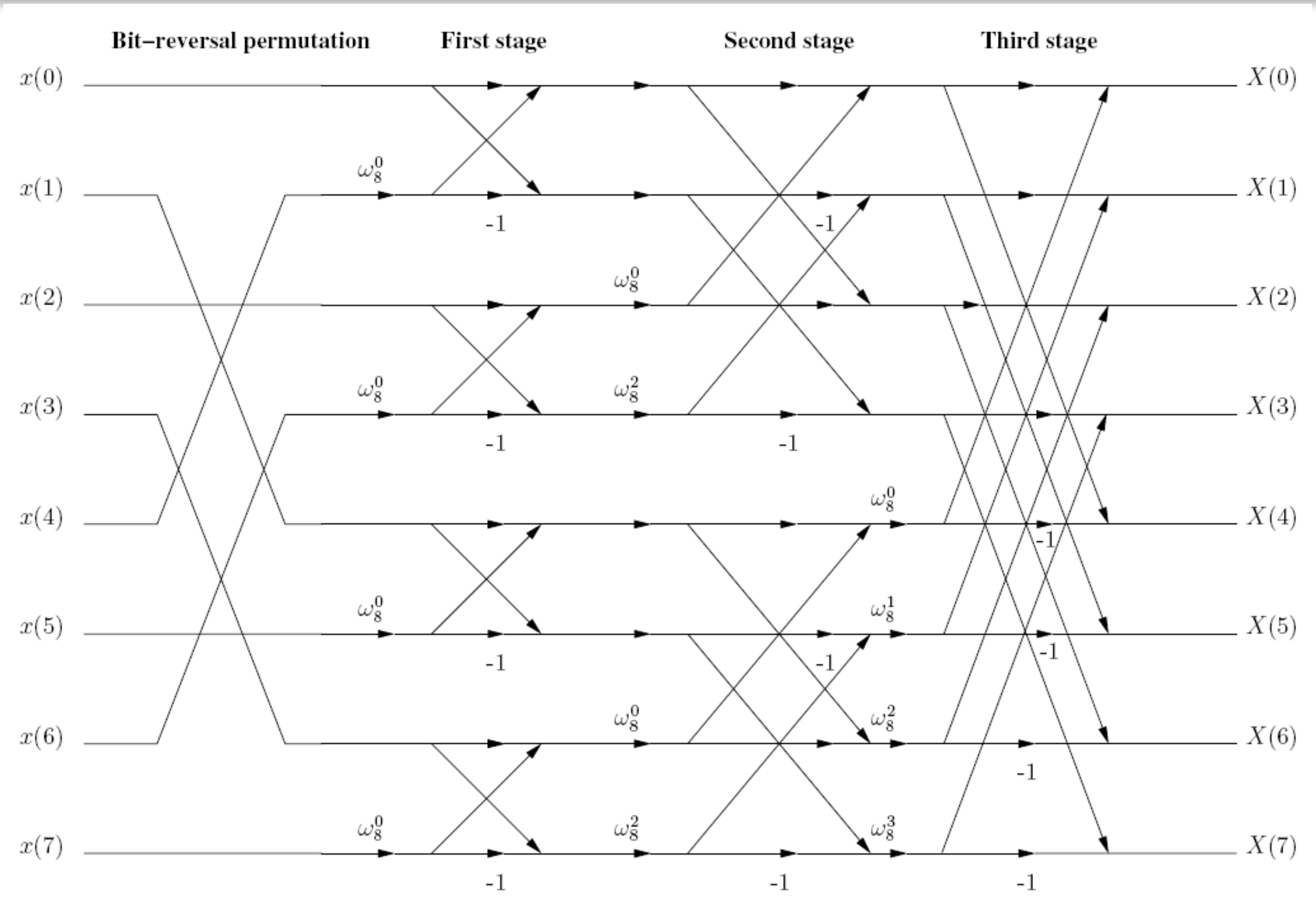
- Radix-2 Cooley-Tukey algorithm: divide and conquer approach.
- Recursively defined



$$\begin{aligned} X(k) &= F_1(k) + \omega_N^k F_2(k), \quad 0 \le k \le \frac{N}{2} - 1 \\ X(k+\frac{N}{2}) &= F_1(k) - \omega_N^k F_2(k), \quad 0 \le k \le \frac{N}{2} - 1 \end{aligned}$$

- w_N^k - twiddle factors, F_i - the N/2-point DFTs of $f_i(n)$.
- Recursive overhead are not favored, iterative implementation are used.

FFT Introduction

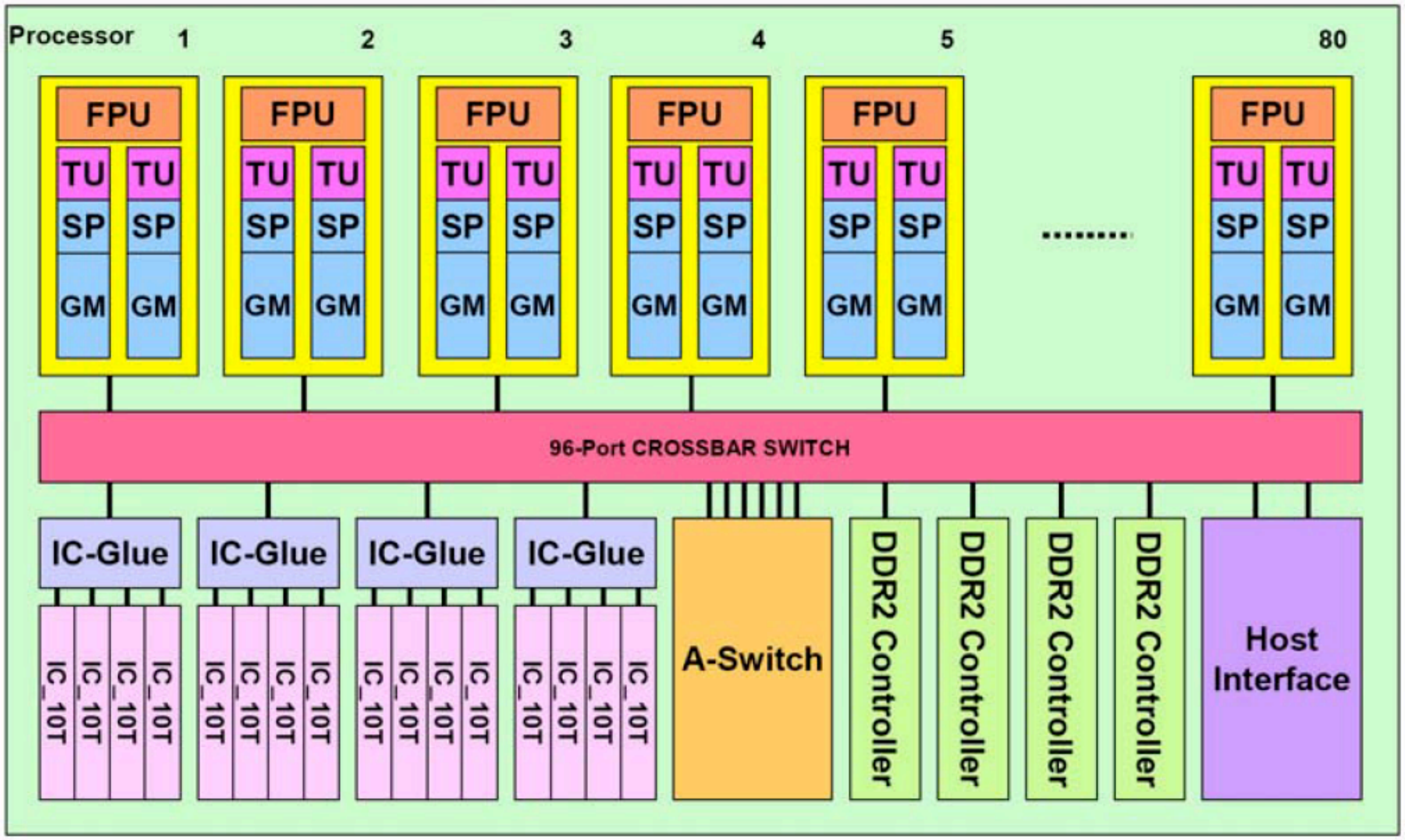


Bit-reversal permutation before butterfly computations

Cyclops-64 Architecture

- Consisting thousands of C64 chips connected by 3D mesh network, with every C64 chip:
  - 80 64-bit processors, each processor 1 floating point unit (FPU) + 2 thread units (TUs).
  - 64 64-bit registers and 32 KB SRAM.
  - 16 shared instruction caches (ICs)
  - 4 off-chip DRAM controllers,
  - Crossbar network with 96*96 ports, 4GB/s bandwidth per port, 384GB/s in total.
  - Memory: scratch-pad (SP) memory, on-chip global interleaved memory (GM), and off-chip DRAM
  - GigaBit Ethernet controller and other I/O devices
  - Etc.

Cyclops-64 Chip

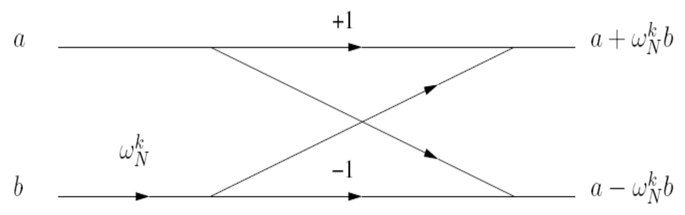


Optimization Analysis – 1D

- Base Parallel Implementation
- Optimal Work Unit
- Special Handling of the First Stages
- Unnecessary Memory Operations
- Loop Unrolling
- Register Renaming and Instruction Scheduling
- Memory Hierarchy Aware Compilation

Base Parallel Implementation

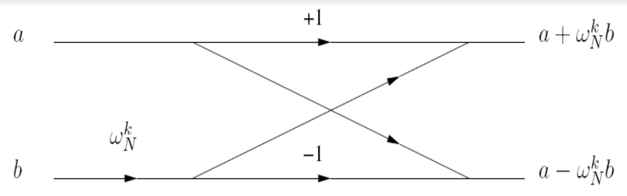
- Work Unit: smallest unit of concurrency.
- Intuitive work unit considers a butterfly operation:
  - Read 2 point data and the twiddle factor from GM
  - Perform a butterfly operation upon them
  - Write the 2 point results back to GM

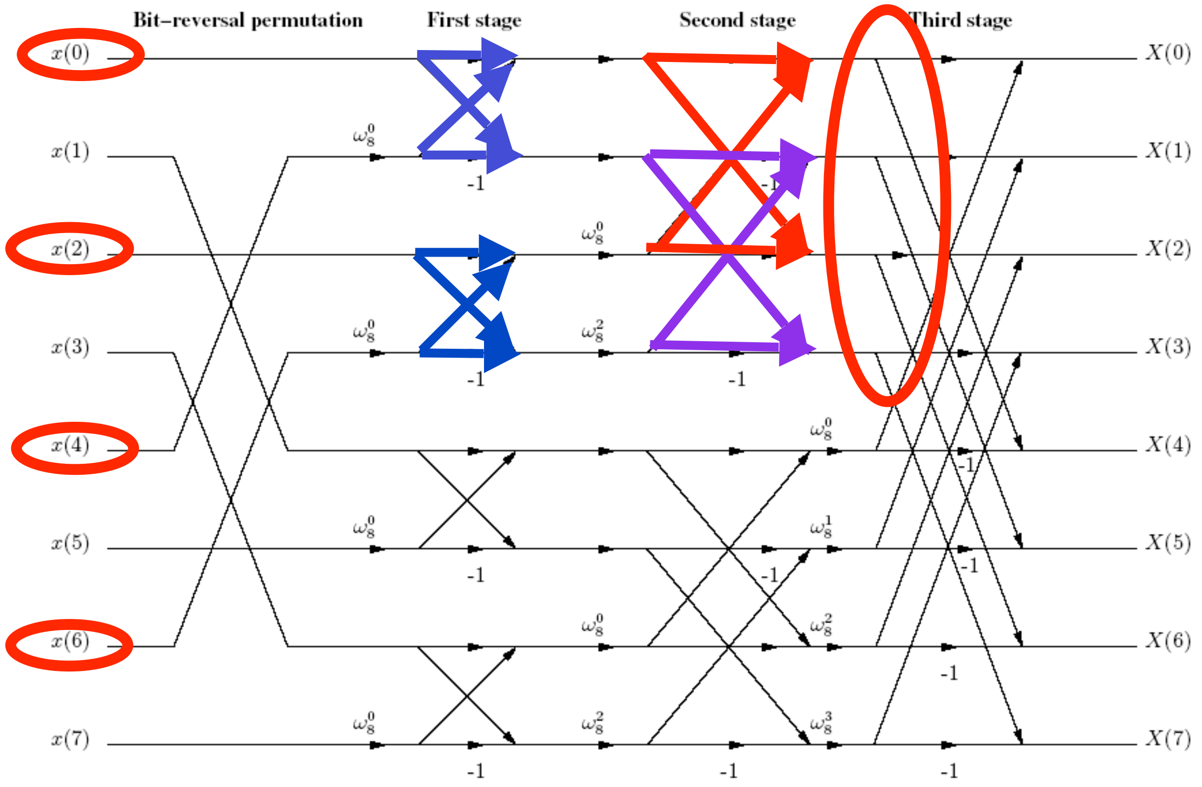


- Work units are assigned in a round-robin way.
- 6.54 Gflops are achieved in this implementation

Butterfly Work Unit

- 1 Butterfly Operation
- 4 Butterfly Operation





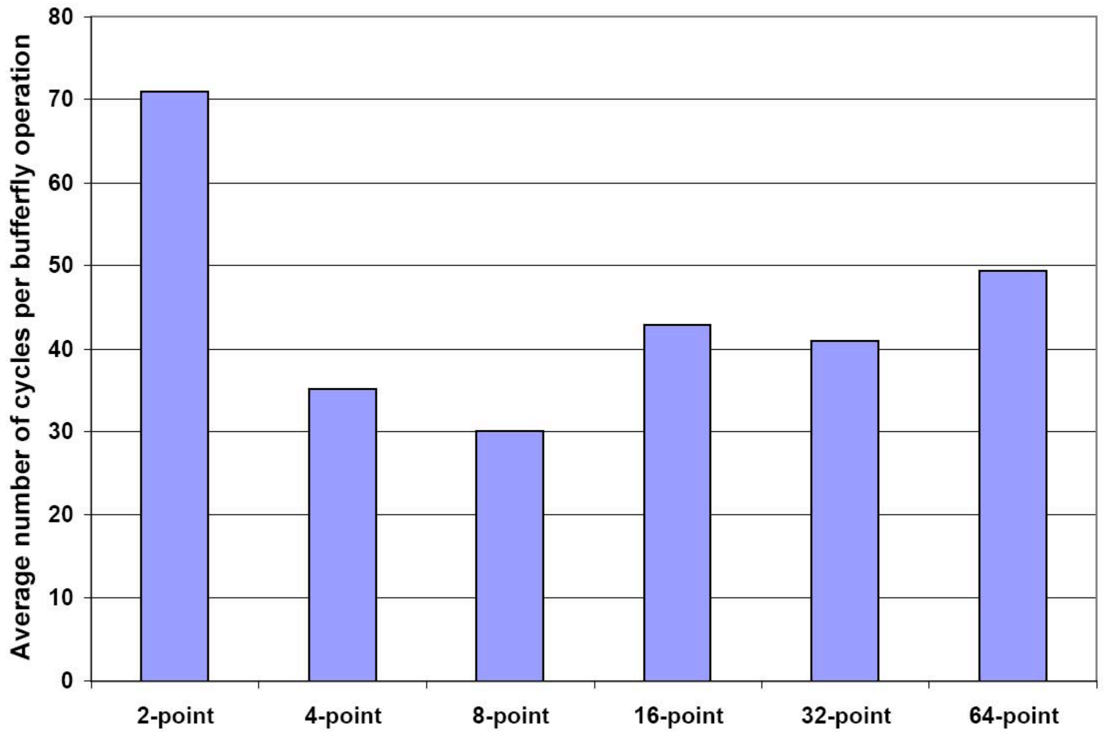
Optimal Work Unit

- Fine-grained work units imply large synchronization overhead.
- Number of floating point operations cannot be reduced -- defined by the FFT algorithm itself.

- Using bigger – point work units:
    - the number of load and store operations are efficiently reduced.
    - the number of stages (number of barriers) are reduced.

Optimal Work Unit

- Number of cycles per butterfly operation VS the the size of work unit (8 point is the best)



- Register spilling for large WU (Need 112 for 16-point)

Optimal Work Unit

- Theoretically, a work unit of N-point data can get rid of (logN-1) barriers.
- Percentage of FP operations is $\frac{5N \lg_2 N}{6N \lg_2 N + 4N}$
- For C64 architecture, 8-point work unit is the best choice without serious register spilling
- Reach a performance 13.17 Gflops.

2^{16} 1D FFT incremental Optimization

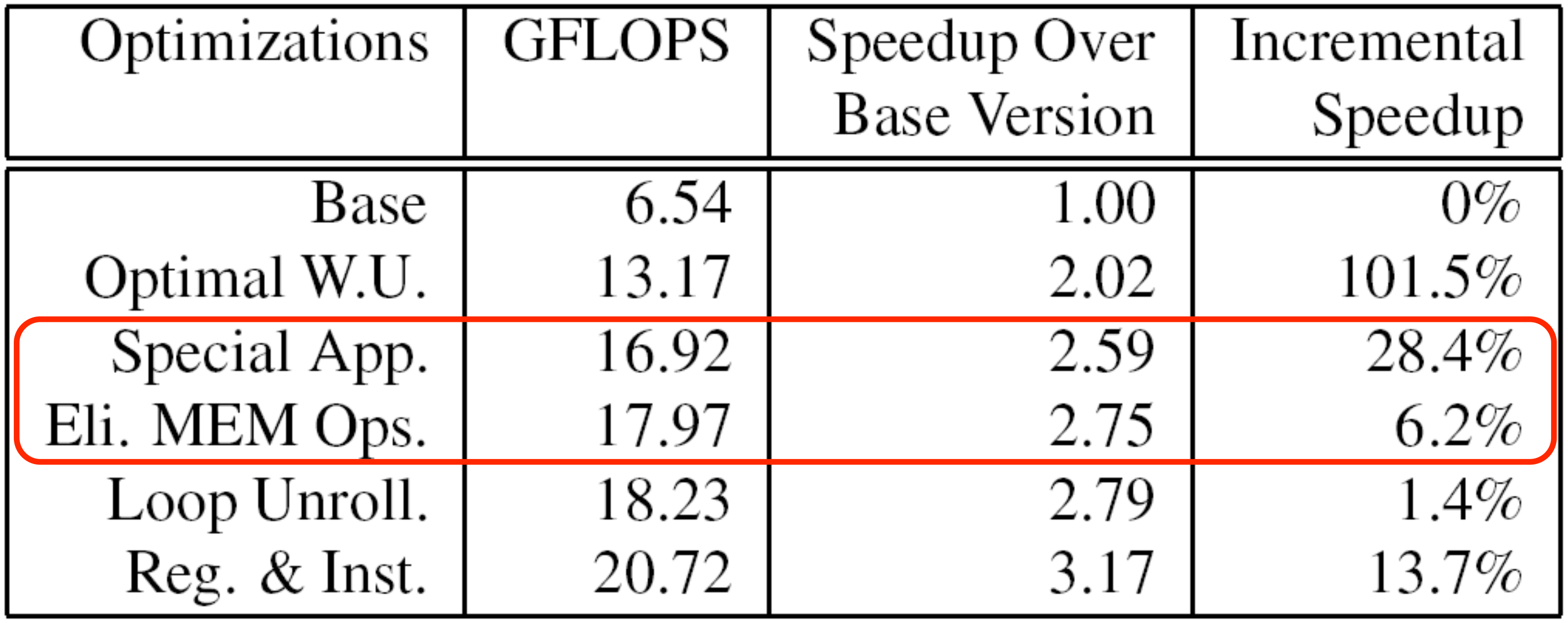
Optimizations	GFLOPS	Speedup Over Base Version	Incremental Speedup
Base	6.54	1.00	0%
Optimal W.U.	13.17	2.02	101.5%
Special App.	16.92	2.59	28.4%
Eli. MEM Ops.	17.97	2.75	6.2%
Loop Unroll.	18.23	2.79	1.4%
Reg. & Inst.	20.72	3.17	13.7%

Thinking about the twiddle factors

- In the first logM stages for M-point work units, all points in the same work unit are consecutive.
- The i-th stage of a complete FFT computation, 2^{i-1} distinct twiddle factors are needed.
- Thus apply 16-point work unit for the first 4 stages, reaching 16.94Gflops.

- Half twiddle factors used in a later stage are the same as those twiddle factors in the previous stage.
- Thus reduce the computation for the indices of twiddle factors and memory operations.

2^{16} 1D FFT incremental Optimization

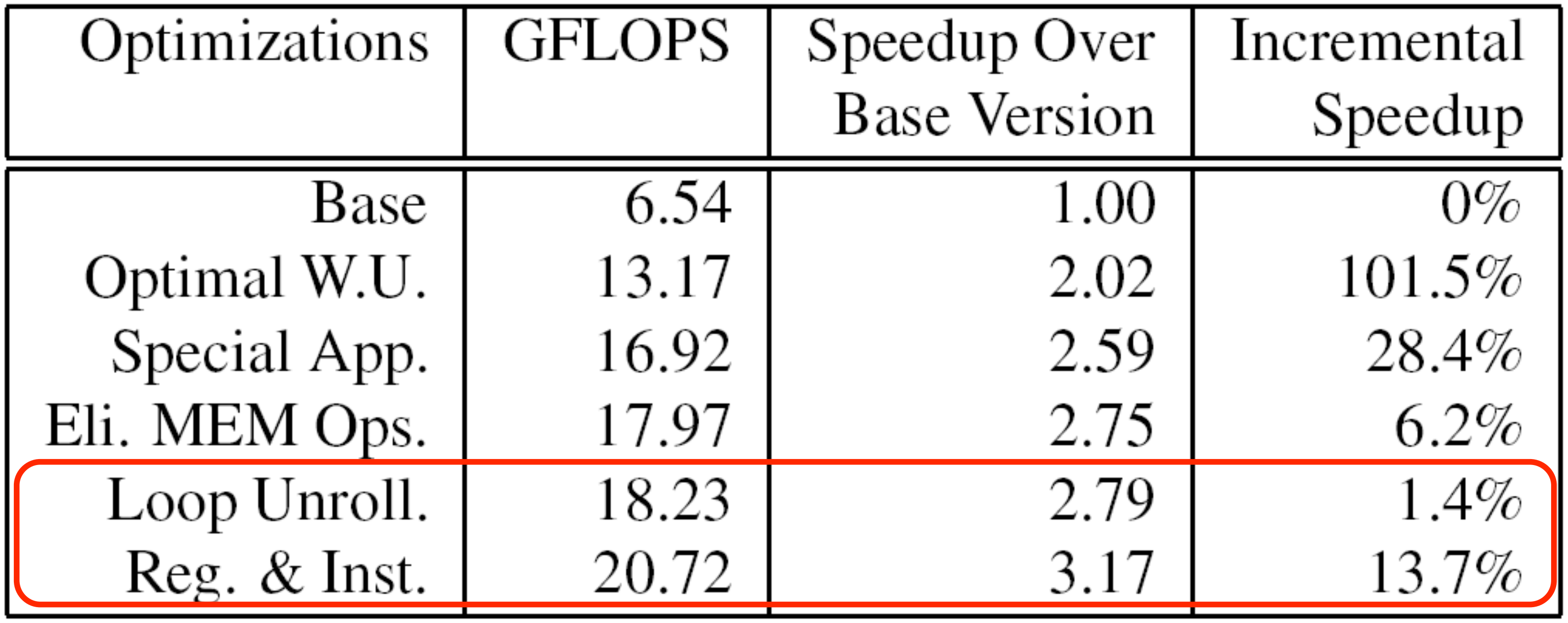


Optimizations	GFLOPS	Speedup Over Base Version	Incremental Speedup
Base	6.54	1.00	0%
Optimal W.U.	13.17	2.02	101.5%
Special App.	16.92	2.59	28.4%
Eli. MEM Ops.	17.97	2.75	6.2%
Loop Unroll.	18.23	2.79	1.4%
Reg. & Inst.	20.72	3.17	13.7%

Loop unrolling & renaming

- Focus on bit-reversal permutation part. (5.7% of total execution time)
- C64 ISA bit gather instruction used to do fast indices computation. Unroll kernel loop 4 times, o hide the memory latency.
- 25% improvement for permutation part, 1.4% improvement on the overall performance.
- Further apply manual renaming and re-scheduling, achieve 13.7% improvement, 20.72 Gflops.

2^{16} 1D FFT incremental Optimization



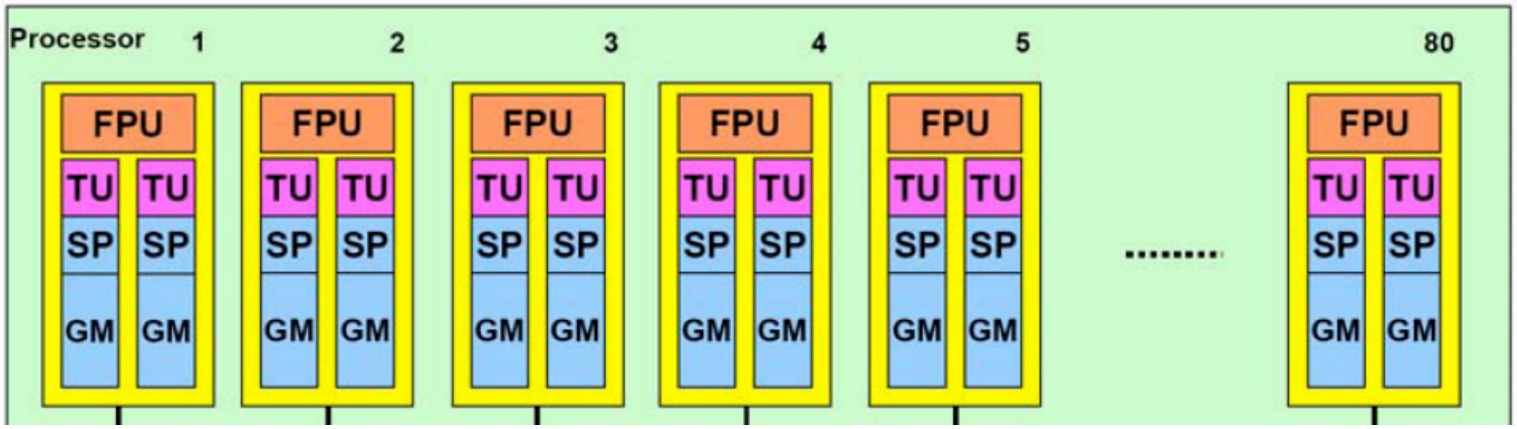
Optimizations	GFLOPS	Speedup Over Base Version	Incremental Speedup
Base	6.54	1.00	0%
Optimal W.U.	13.17	2.02	101.5%
Special App.	16.92	2.59	28.4%
Eli. MEM Ops.	17.97	2.75	6.2%
Loop Unroll.	18.23	2.79	1.4%
Reg. & Inst.	20.72	3.17	13.7%

Memory Hierarchy Aware Compilation

- Entire process is tedious and error-prone.
- Smart compiler: identify the segments where variables reside, apply corresponding latencies when scheduling the instructions.
- 19.84Gflops using tailored compiler on loop unrolled code.

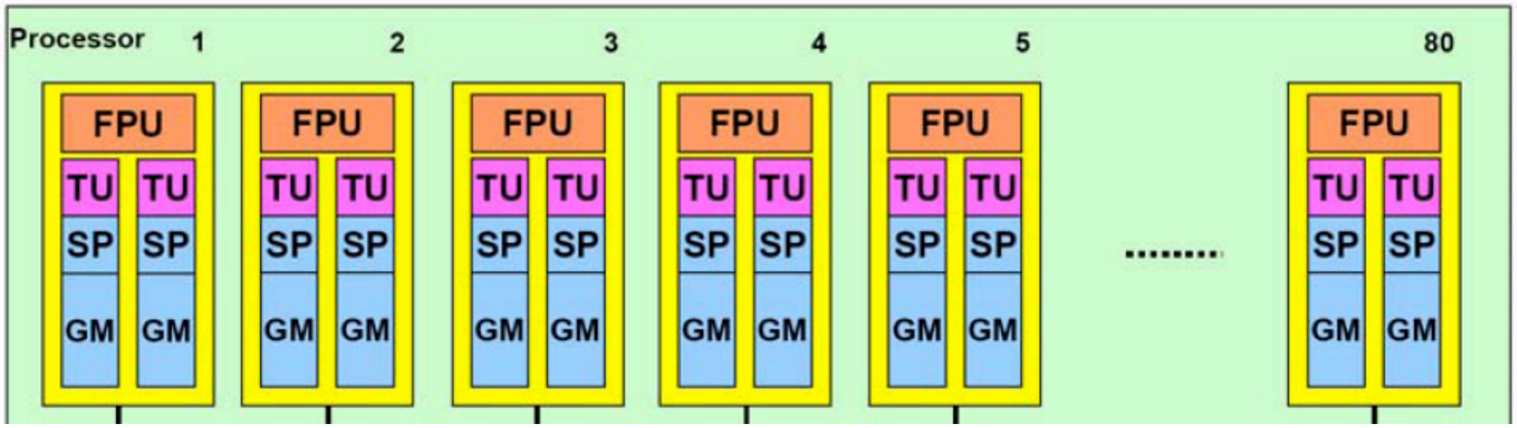
2D FFT

- Perform 1D FFT alternatively on each dimension of the data interleaved with data transpose steps.
- One row/column FFT as a work unit.
- Every row/column are independent to each other, work units are distributed to threads in the round-robin way.
- 15.11Gflops achieved.



- Some threads remain idle (e.g. 180 rows, 160 threads)

Load Balancing



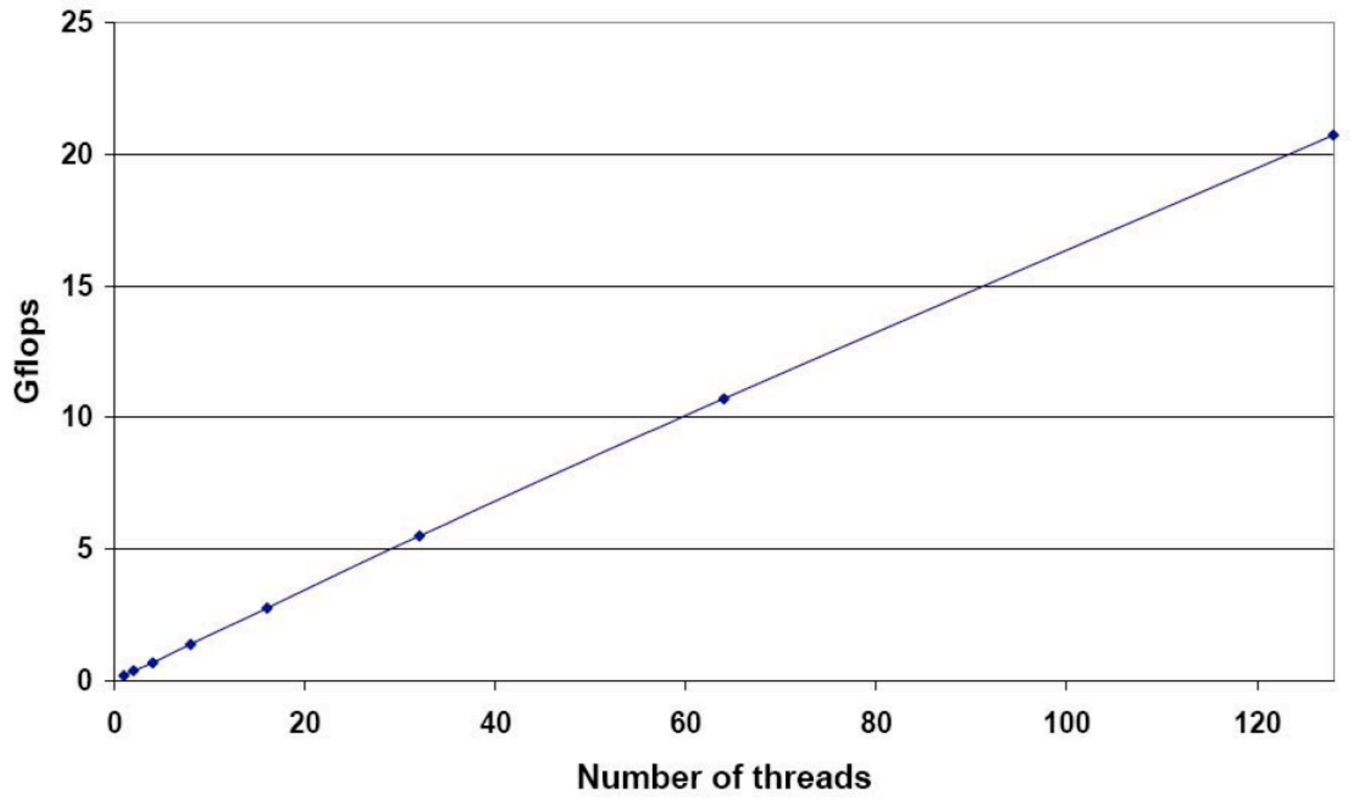
- Base parallel implementation straightforward, but not necessarily efficient.
- Not fine enough grained, using smaller work unit instead
- Small task: 8-point work unit. (8 input<-> 8 output)
- it needs more barriers to synchronize threads working on the same row/column FFT

Work Distribution and Data Reuse

- Exploit the nature of 2D FFT: exact the same operations and twiddle factors are applied on each row/column FFT.
- This character favours data reuse, which can reduce indices computation and memory operations.
- Major-reversal work distribution scheme to exploit this opportunity, 19.37Gflops achieved.

Speedup of optimized FFT

- 1D FFT   2^16 points and 2D FFT 256*256



Conclusion

- Conclusion:
  - Consider both the architecture features and application characteristics.
  - A set of optimization techniques are proposed. (Essentiality: reduce memory operation)
  - Challenges to multi-core system software: smart compiler.
  - Achieve 20Gflops on both 1D and 2D FFT, which is about 4 times of Intel Xeon Pentium processor (about 5Gflops).
- Future work:
  - Fast scratchpad memory on thread unit may be used as larger register file. Larger point work unit may be exploited.
  - Larger FFT problem size when data cannot be fully stored.

Questions?

.
.
.
.
.
.
.
.
.
.

Thanks for your time…