Efficient Implementation of Sorting on Multi-Core SIMD CPU Architecture

Jatin Chhugani

William Macy

Akram Baransi

Anthony D. Nguyen

Mostafa Hagog

Sanjeev Kumar

Victor W. Lee

Yen-Kuang Chen

Pradeep Dubey

Agenda

- Paper Objective
- Architecture Specs and Algorithmic Impacts
- SIMD 101
- Paper Implementation
- Analysis
- Result

Contribution

- Fastest sorting performance for modern computer architectures
- Implementation of the Multi-Way Merge
- Avoiding expensive unaligned load/store ops

Developments in Architecture

- Evolution of different features over time that have brought a substantial performance improvement in algorithmic implementation
- Parallelism at different levels

1. ILP
2. DLP
3. TLP
4. MLP

ILP

- Instruction Level Parallelism eg. Branch prediction, out of order execution

DLP

- Data Level Parallelism
- SIMD

TLP

- Thread level parallelism eg. hyperthreading
- Relies on cache design

MLP

- Memory level parallelism eg. Hardware prefetchers

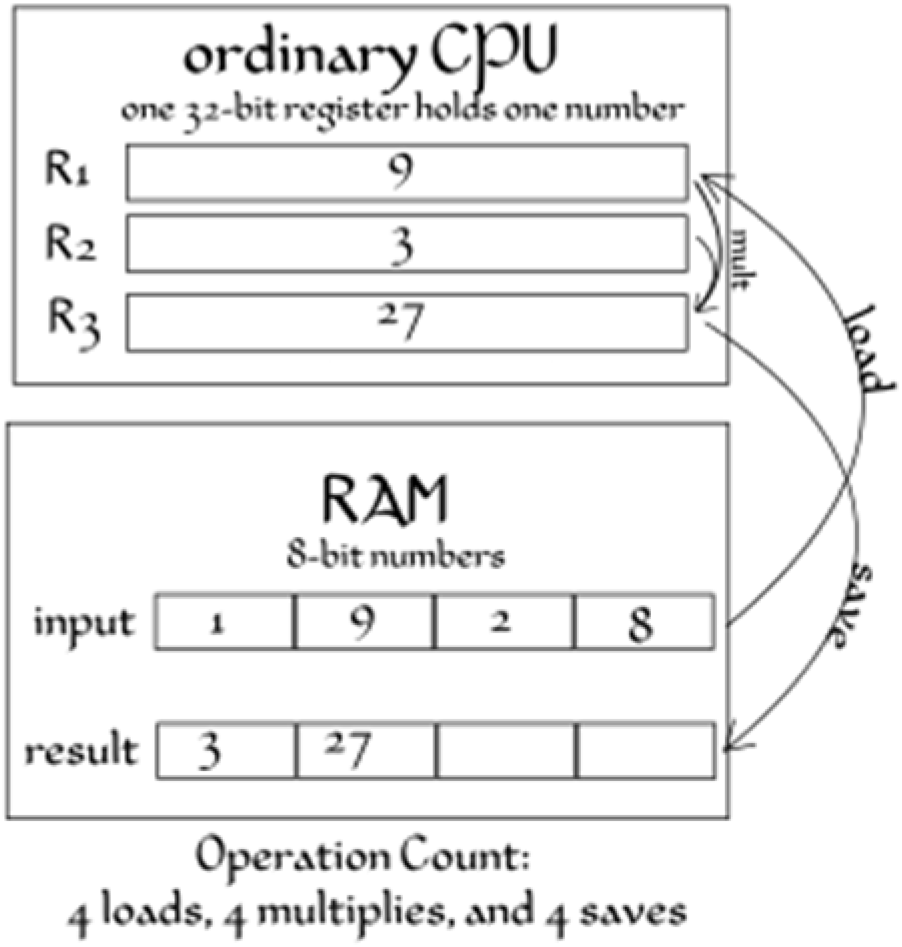
SIMD (Sim – D)

- Single Instruction Multiple Data

MMX Registers

SSE

- Streaming SIMD Extensions
- Enhanced Instruction Set
- Accounts for both types of registers



SIMD Advantages/Disadvantage:

A.Does uniform computation extremely quickly

D. Not applicable to non-uniform transformations

For advanced computation, we need a *shuffle* operation.

A1 A2 A3 A4, B1 B2 B3 B4 – we want to compare adjacent elements.

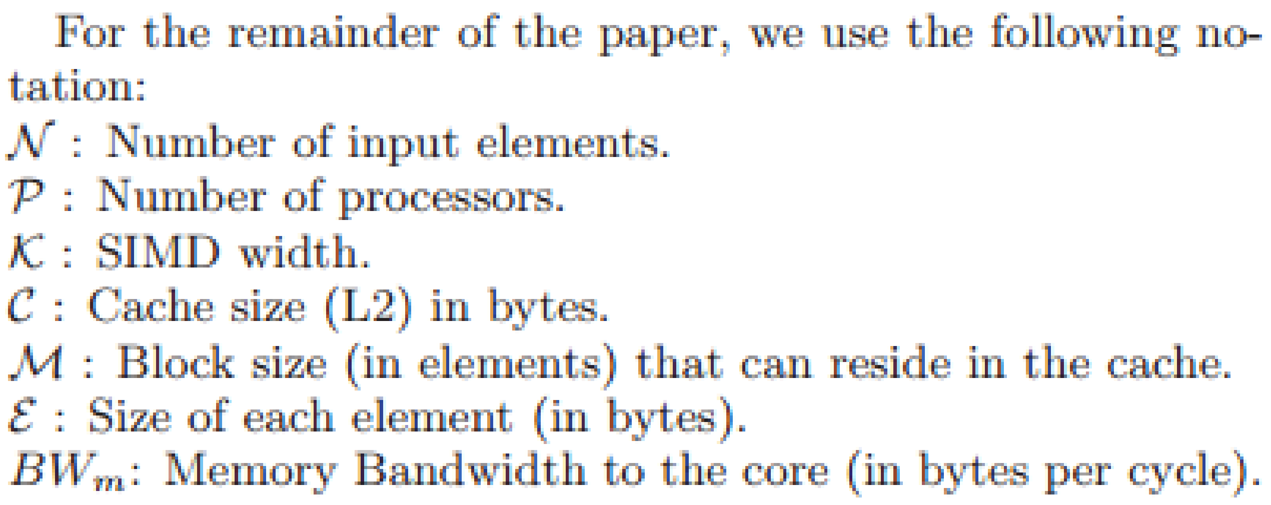
SIMD Instructions

- ADDPS - Add Packed Single-Precision FP Values
- ADDSS - Add Scalar Single-Precision FP Values

- ANDPS: Bitwise Logical AND For Single FP

- CMPccPS: Packed Single-Precision FP Compare
- MOVAPS: Move Aligned Packed Single-Precision FP Values
- MOVUPS: Move Unaligned Packed Single-Precision FP Values

Mergesort and the Proposed Algorithm



For the remainder of the paper, we use the following notation:
$\mathcal{N}$: Number of input elements.
$\mathcal{P}$: Number of processors.
$\mathcal{K}$: SIMD width.
$\mathcal{C}$: Cache size (L2) in bytes.
$\mathcal{M}$: Block size (in elements) that can reside in the cache.
$\mathcal{E}$: Size of each element (in bytes).
BW_m: Memory Bandwidth to the core (in bytes per cycle).

Idea: Take advantage of cache by dividing dataset into chunks
If Cache size is C, Block size = C/2*E

Algorithm

Divided into 2 phases

I

1.Divide input data into blocks of size C/2E

- Merging networks

2. P sorted lists (P threads) merged together

End of phase 1:N/M sorted blocks of size M

Algorithm - ctd

II

1.log(N/M) iterations of merging sorted lists

2.Use multi-way merging to optimize and avoid memory bandwidth issues

Merging Networks

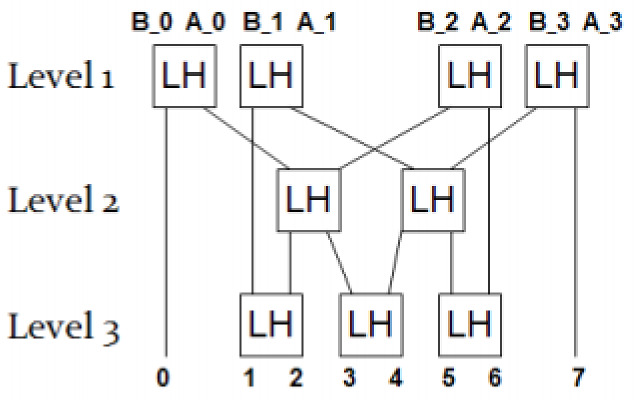


Figure 1: Odd-Even merge network for merging sequences of length 4 elements each.

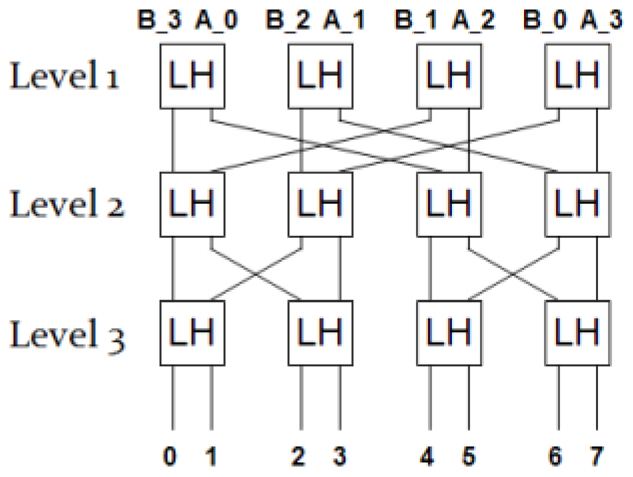


Figure 2: Bitonic merge network for merging sequences of length 4 elements each.

Merging 2 sorted lists

- 2 sorted lists X,Y (length L)
- Assume 4 wide SIMD
- Sorting code runs (2L/4 -1) times

- Latency between operations is a major problem
- Min/Max operation takes 3 cycles, shuffle takes 1 cycle.

Solutions

1. Using a wider network (16x16 bitonic)

2.5X speedup

2. Simultaneously merge multiple lists

3.25X speedup

Exploiting Multiple Cores

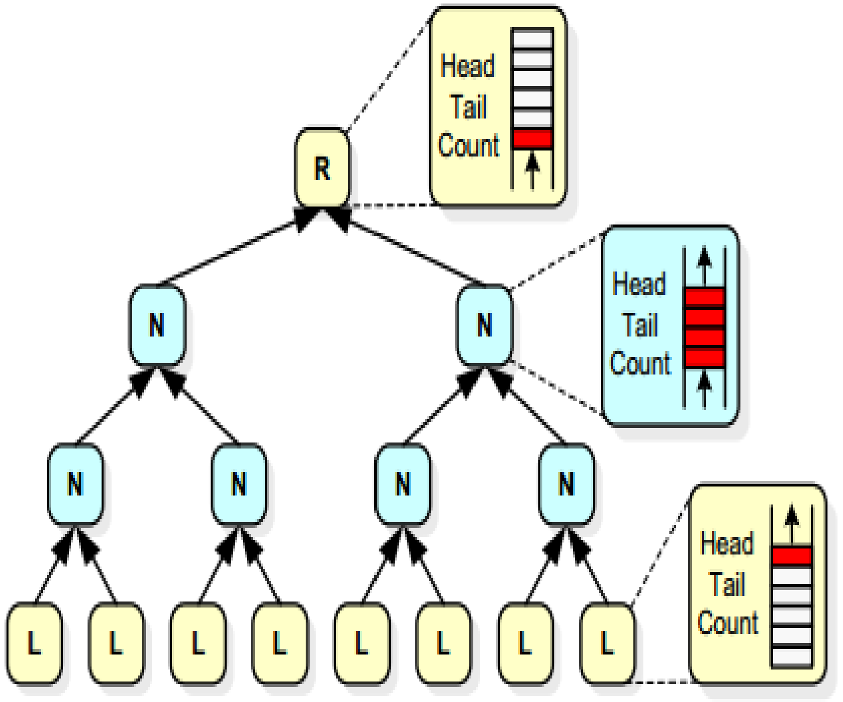
- Bandwidth becomes the new bottleneck
- Multiple threads can share the last-level caches
- Divide the M-size blocks among the P threads, M’ = M/P
- For each pair of consecutive threads, find the median of their merged list.
- Median points may not be aligned locations

MultiWay Merge

Compute individual lists and merge using a binary tree

At any node, if there is an empty slot in its FIFO queue and each of its children has at least one element in its queue, smaller of the two elements is moved into the node's queue from the child's queue.

Process these 'ready' nodes in a 4K by 4K network in parallel.



Analysis

- <u>**Single Thread Scalar Implementation**</u>

Using cache, we can ignore the memory access time for one block compared to the computation of another.

$$\mathcal{T}_{serial-phase1} = \mathcal{N} * \log(\mathcal{M}) * \mathcal{T}_{se}$$

$$\mathcal{T}_{serial-phase2} = \mathcal{N} * \mathrm{Max}(\log(\mathcal{N}/\mathcal{M})\ \mathcal{T}_{se},\ 2\mathcal{E}/BW_m)$$

• Single Thread SIMD Implementation

Let a be the min/max latency, b be the shuffle latency, c be the communication latency/inter functional unit latency.

We want to find total running time based on these and K.

Generalized expression:
((a+b+2c+2)log2K + (b+c+22))/4K = 3 cycles/element on a 4 wide SSE

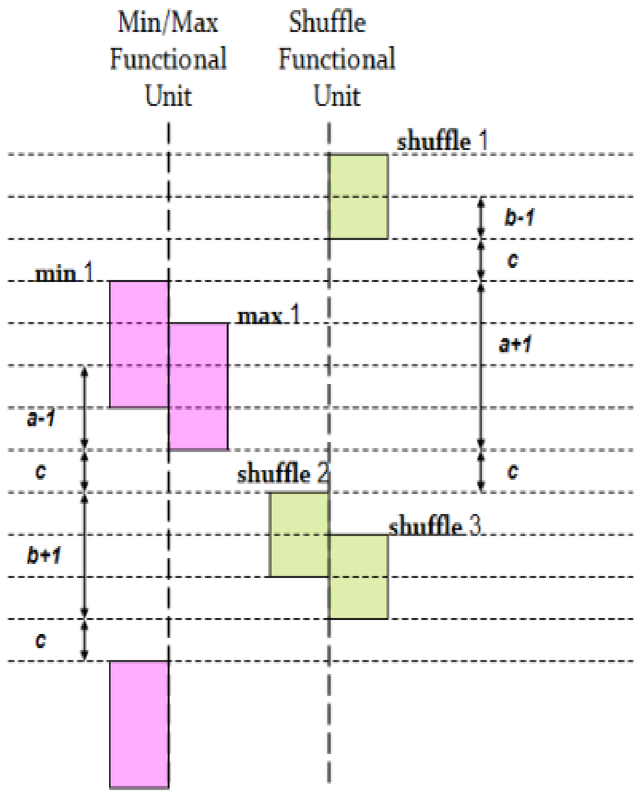


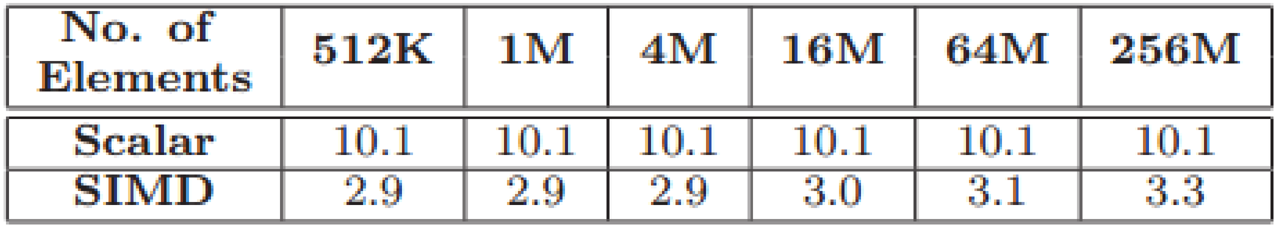
Figure 7: Timing diagram for the two functional units during the execution of the merging network.

- Time barrier for synchronization is proportional to the number of cores

$$\mathcal{T}_{parallel-phase1b} = \log(\mathcal{P}) * [\mathcal{M}\mathcal{T}_{pe} + \mathcal{T}_{sync}]$$

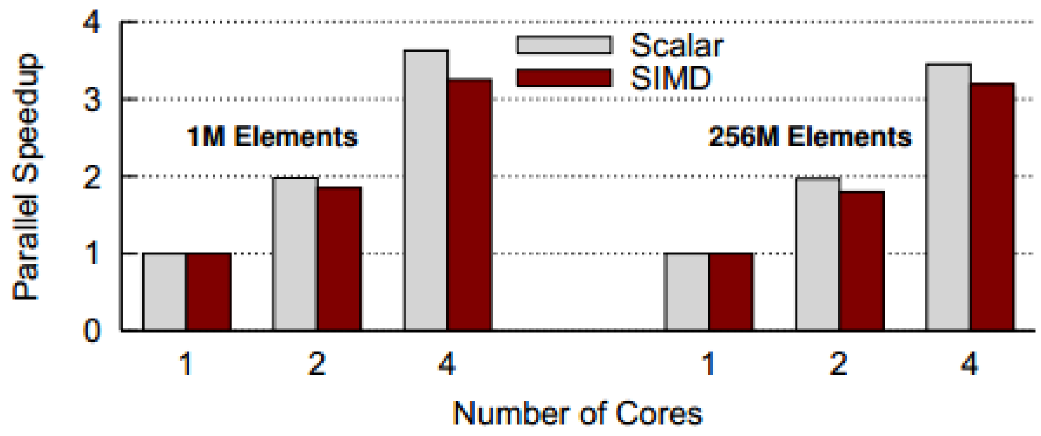
$$\mathcal{T}_{parallel-phase2} = \log(\mathcal{N}/\mathcal{M}) * [\mathcal{N}\text{Max}(\log(\mathcal{N}/\mathcal{M})\mathcal{T}_{pe}, 2\mathcal{E}/BW_m) + \mathcal{T}_{sync}]$$

Results



No. of Elements	512K	1M	4M	16M	64M	256M
Scalar	10.1	10.1	10.1	10.1	10.1	10.1
SIMD	2.9	2.9	2.9	3.0	3.1	3.3

Table 3: Single-Thread Performance (in cycles per element per iteration).


Parallel Speedup
Scalar
SIMD
1M Elements
256M Elements
Number of Cores