Parallelizing Sorting Algorithms

Mark Adams, Ksenia Burova
EECS

THE UNIVERSITY OF TENNESSEE KNOXVILLE

April 5th, 2018

Questions

1. How much work Even-Odd sort does if there are *n* elements in a set?
2. Who invented first electromechanical punched card tabulator?
3. What are two conditions to watch out for when using MPI?

Who are we?

Name: Ksenia Burova
Degree: 5-yr BS/MS in CS
Hometown: Vladivostok, Russia
Interests: cats, gym, aerial sports (silks, pole, hoop)

Who are we?

Name: Mark Adams
Degree: PhD, CS
Hometown: Naperville, IL
Research Interests: HPC, Scientific computing and
Big Data, ML, and AI

Work: Full-time R&D Staff at ORNL

OAK RIDGE National Laboratory | BUILDING TECHNOLOGIES RESEARCH AND INTEGRATION CENTER

CTO of Tunation, LLC

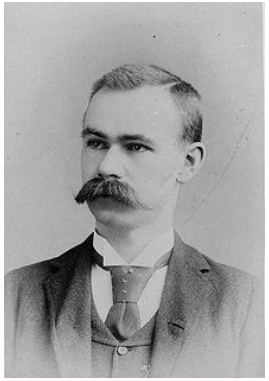
Fun facts: Hat trick of degrees
Middle school in Supersize Me
2 different martial arts during BS and MS

Outline

- History of sorting
- Sequential algorithms overview
- Motivation and applications
- GPU/CUDA overview
- Parallel algorithms for GPU
- OpenMP, MPI overview
- Parallel algorithms for distributed systems
- Open issues
- Discussion

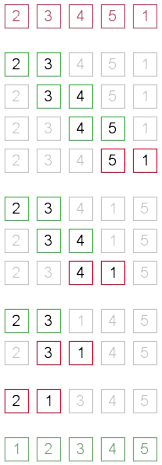
History of Sorting

- First machines for sorting - 19th century
- First electric tabulating machine (1890)
  - Herman Hollerith
  - Improved version, basis for radix sort (1901-1904)
  - Tabulating Machine Company (1896)
- The collator (1938)
  - Idea of merging employed
  - James W. Bryce (IBM)
- Sort was the first routine written for a stored program computer

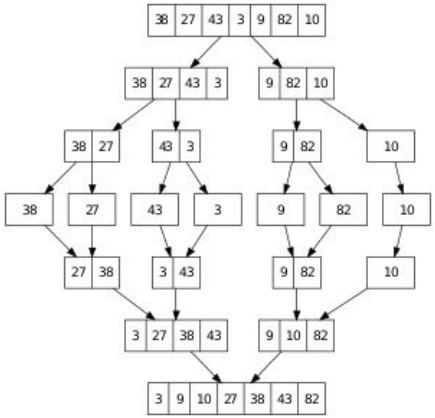


Algorithms Overview

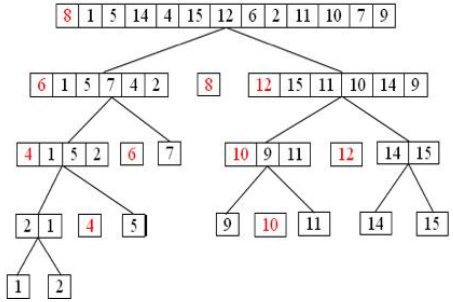
Bubble Sort



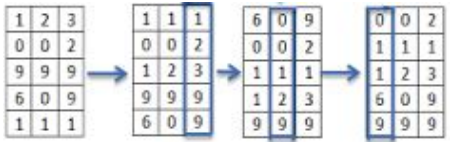
Merge Sort



Quick Sort



Radix Sort



Motivation and Applications

- **Sorting** is a fundamental problem in computer science
- There is a limit on the efficiency
- Motivation
  - Growth of data
  - Hard to achieve significant increases in speed by using faster devices
  - Parallelism as alternative route to attain high computational speed
- Applications
  - Software development, computer organization and systems design
  - Data processing

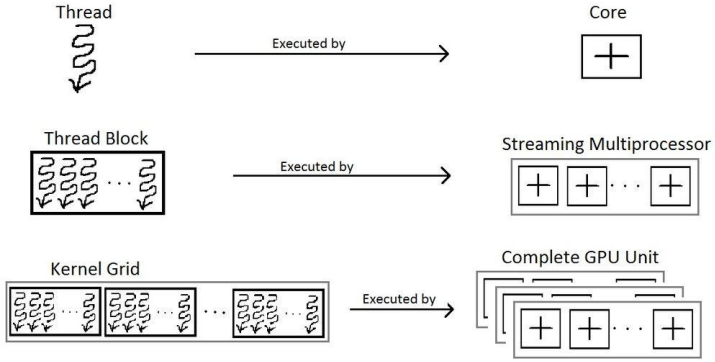
GPU / CUDA Overview

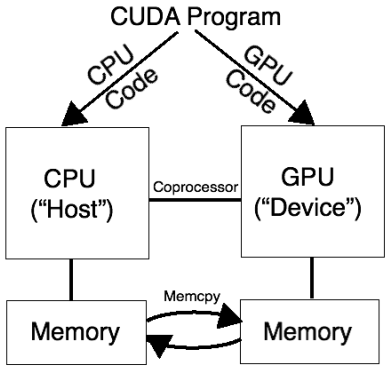
- GPU (graphics processing unit)
  - Lot of simple control units
  - Trade simple control for more compute
  - Explicitly parallel programming model
  - Optimize for throughput not latency

- CUDA
  - parallel computing platform that HLL can exploit for parallelism

- Programming Model
  - Kernel - C/C++ function
  - Threads - paths of execution
  - Thread blocks - group of threads that cooperate to solve a subproblem

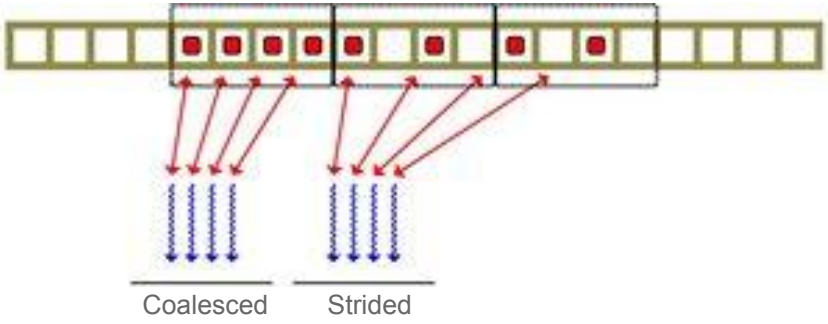
GPU is responsible for allocating blocks to the SMs!





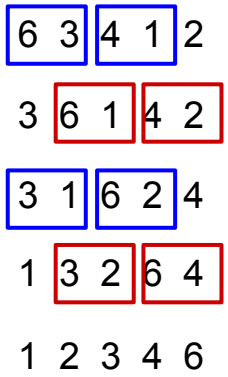
GPU Parallel Algorithm

- Keep hardware busy (a lot of threads)
- Limit branch divergence (if-else can’t be executed concurrently)
- Prefer coalesced memory access



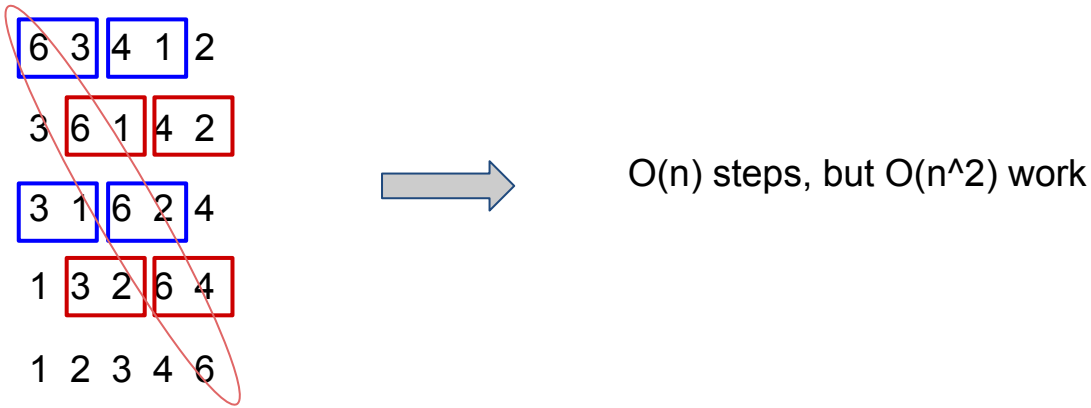
Even-Odd Sort

- Parallel version of Bubble sort (Brick sort)
- Maps nicely to a parallel implementation



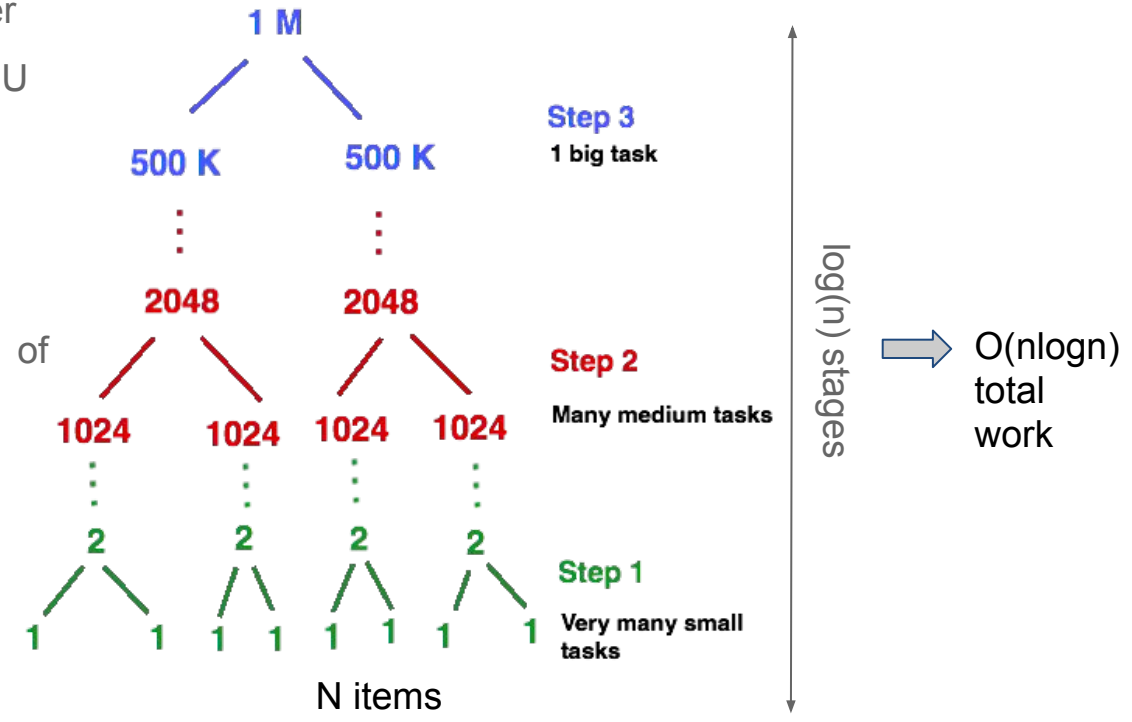
Even-Odd Sort

- Parallel version of Bubble sort
- Maps nicely to a parallel implementation



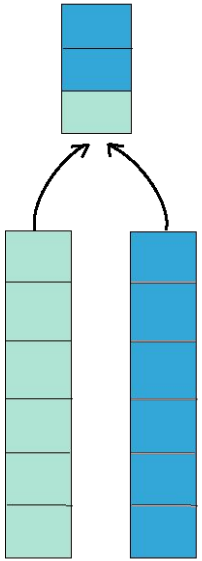
Merge Sort

- Great instance of divide and conquer
- Really interesting how to map to GPU
  - Bottom-Up approach
  - Start with many tiny problems
  - End up with one large
- Hard to map due to different sizes of subproblems

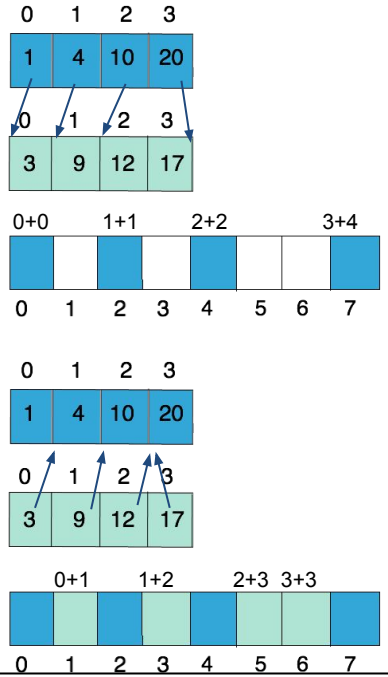


Merge

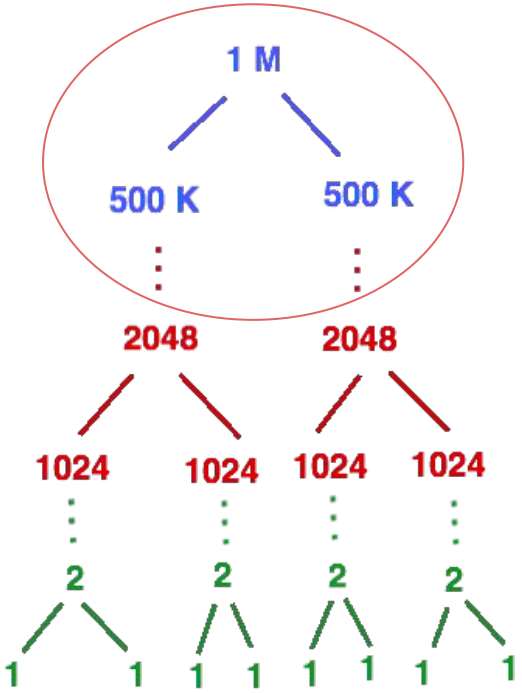
Sequential



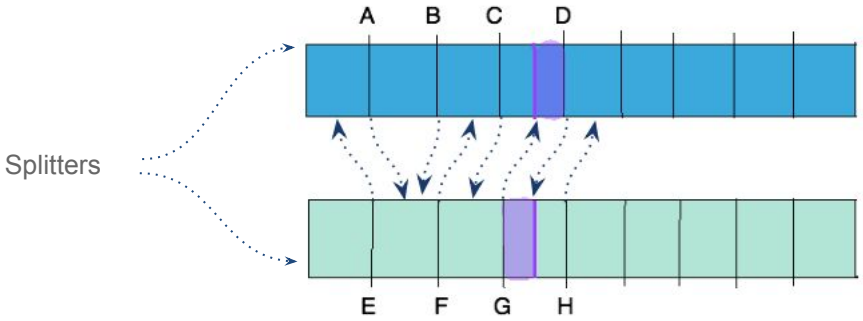
Parallel

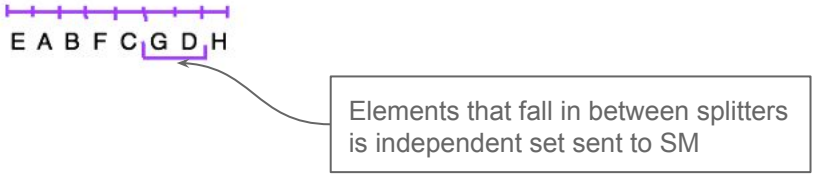


Merge Sort



- Split large one large task across SMs
  - Split longer lists into shorter sublists
  - Use splitters (every Nth element in each array)

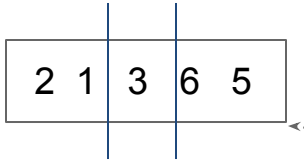


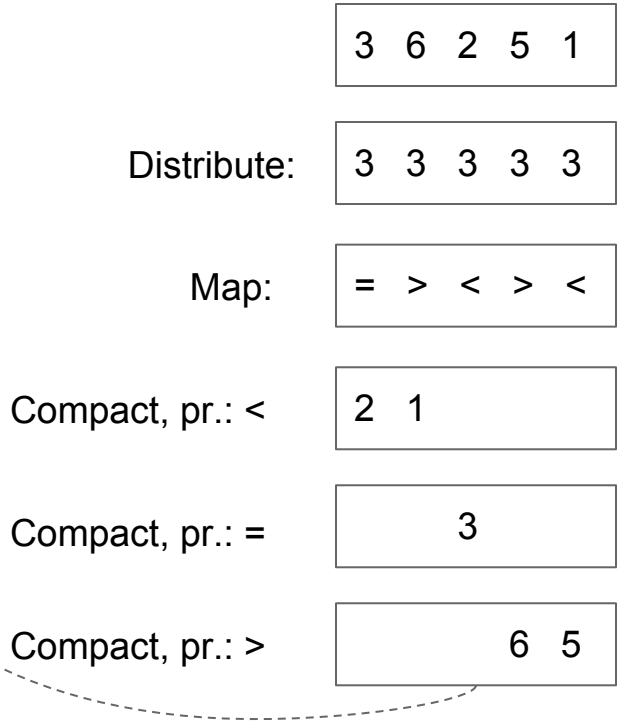


Quick Sort

- Most efficient for serial processors
- Hard to implement on GPU because of control complexity
  - Recursion
  - Different size of subarrays

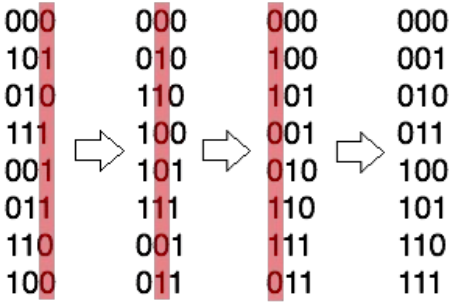
- Eliminate recursion using segments
  - **Distribute** (Seg.) pivot across a segment
  - **Map** to split a segment
  - **Compact**
  - Hard to implement

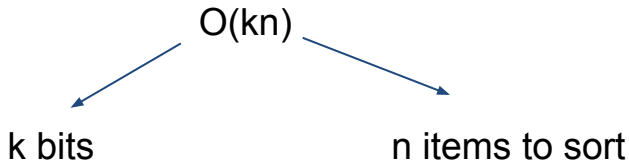




Radix Sort

- Start LSB
- Split input into 2 sets based on bit. Otherwise preserve order
- Move to next MSB. Repeat.





Underlying split operation is the one that is can be done efficiently !

Radix Sort

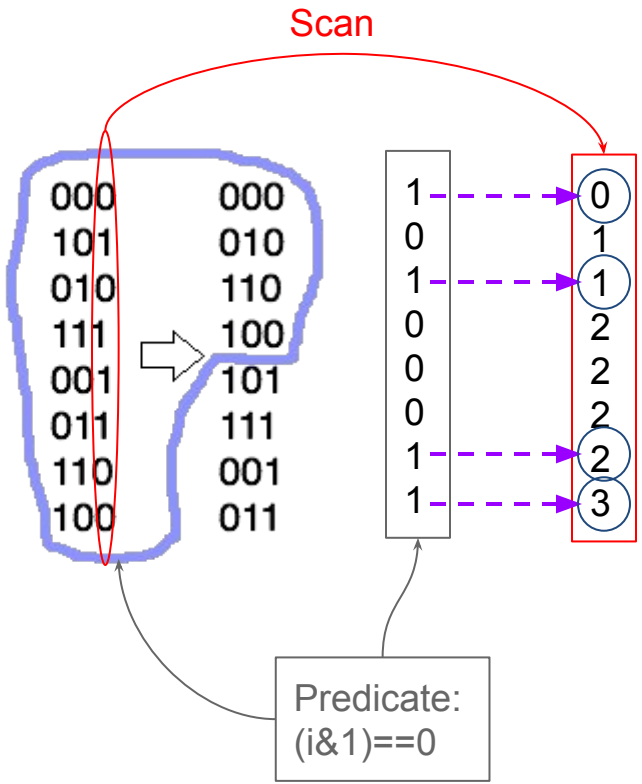
- Use **Compact** algorithm
  - “Filter”, leave items we care about

Input:	s0	s1	s2
Predicate:	T	F	T
Output:	s0	--	s2

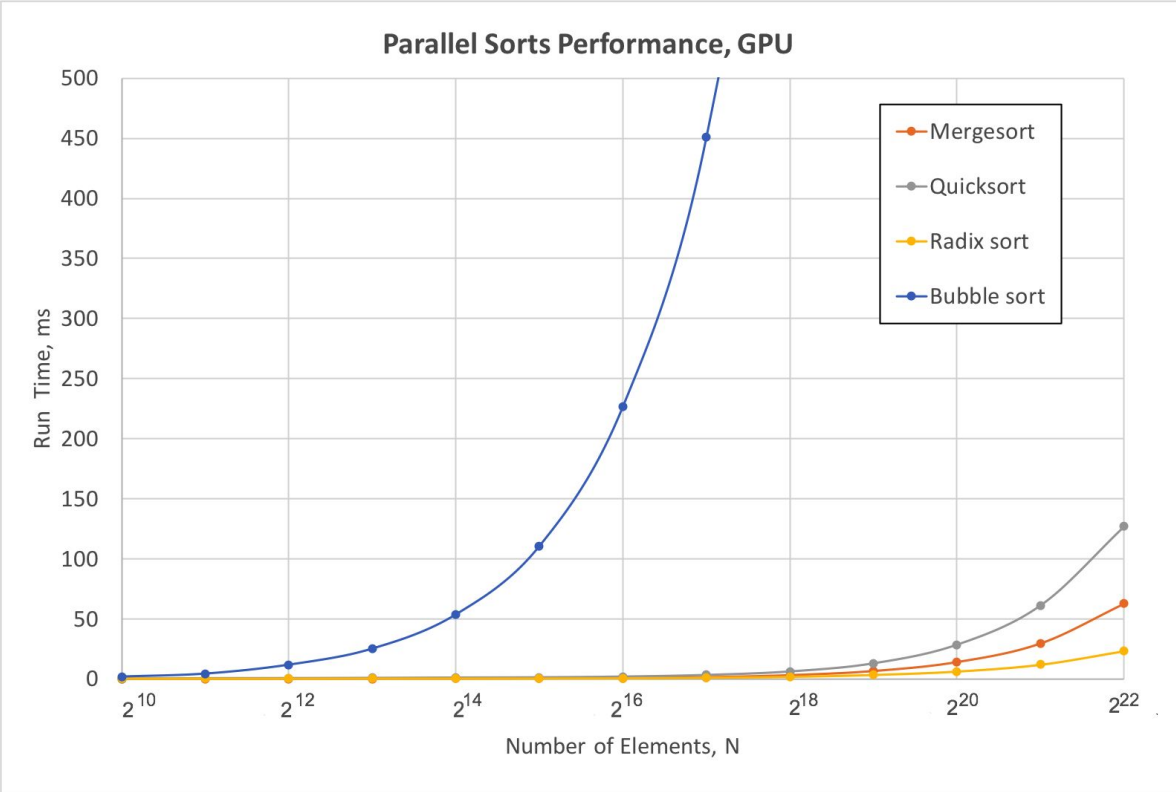
- Use **Scan** algorithm

Input:	1	2	1	4
Op:	ADD			
Output:	1	3	4	8

  - Useful only for parallel implementation
  - Used to determine index



Performance



Message Passing Interface (MPI)

1992 - Preliminary draft (Jack Dongarra, Tony Hey, and David W. Walker)

1993 - Draft MPI standard

1994 - MPI 1.0 release

1997 - MPI 2.0 release

2012 - MPI 3.0 release

Jack Dongarra

UTK EECS - University Distinguished Professor

Innovative Computing Lab - Director

ORNL - Distinguished Research Staff

All parallelism is explicit: the programmer is responsible for correctly identifying parallelism and implementing parallel algorithms using MPI constructs

Message Passing Interface (MPI) - Hello World!

```
int main(int argc, char** argv) {
  int world_size, world_rank, name_len;
  char processor_name[MPI_MAX_PROCESSOR_NAME];

  MPI_Init(NULL, NULL);
  MPI_Comm_size(MPI_COMM_WORLD, &world_size);
  MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
  MPI_Get_processor_name(processor_name, &name_len);

  printf("Hello world from %s, rank %d/%d\n", processor_name, world_rank, world_size);

  MPI_Finalize();
}
```

Message Passing Interface (MPI)

Move data from one process to another (possibly local) process

– The data is described by a data-type, a count, and a memory location

– The destination process by a rank in a communicator

– The matching is tag based

int MPI_Send(void* buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)

int MPI_Recv(void* buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status* status)

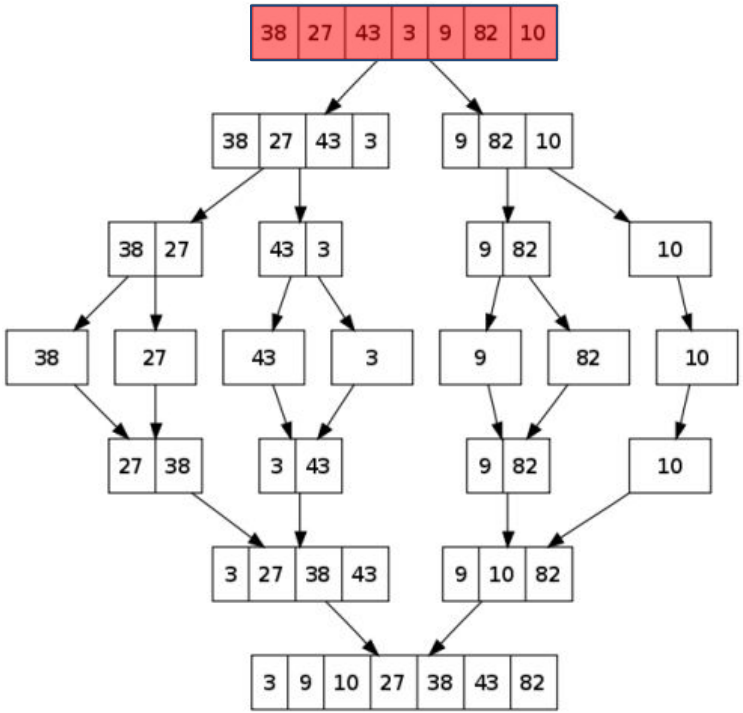
Be careful of ***blocking*** and ***race conditions*** when communicating

Parallel MPI Merge Sort

- Similar to GPU-based approach

- Recursively sub-divide problem across all available nodes
  - Send 2nd half to another processor
  - Worker processor starts traversing its leaf
  - When complete, sends sorted array to parent
  - Parent receives sorted array then merges with sorted 1st half.

- Once out of available processes, run serial mergesort
- When array sizes are small, use insertion sort instead

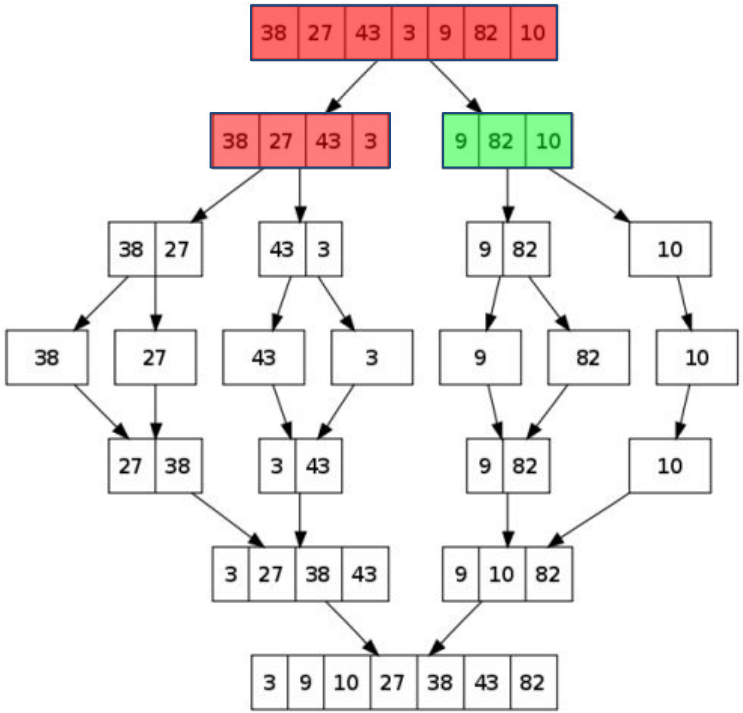


Parallel MPI Merge Sort

- Similar to GPU-based approach

- Recursively sub-divide problem across all available nodes
  - Send 2nd half to another processor
  - Worker processor starts traversing its leaf
  - When complete, sends sorted array to parent
  - Parent receives sorted array then merges with sorted 1st half.

- Once out of available processes, run serial mergesort
- When array sizes are small, use insertion sort instead

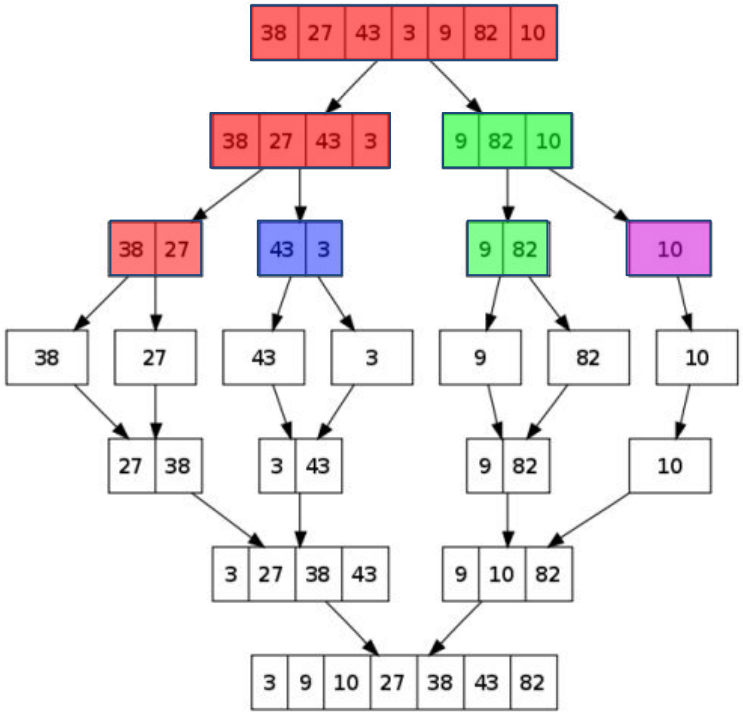


Parallel MPI Merge Sort

- Similar to GPU-based approach

- Recursively sub-divide problem across all available nodes
  - Send 2nd half to another processor
  - Worker processor starts traversing its leaf
  - When complete, sends sorted array to parent
  - Parent receives sorted array then merges with sorted 1st half.

- Once out of available processes, run serial mergesort
- When array sizes are small, use insertion sort instead

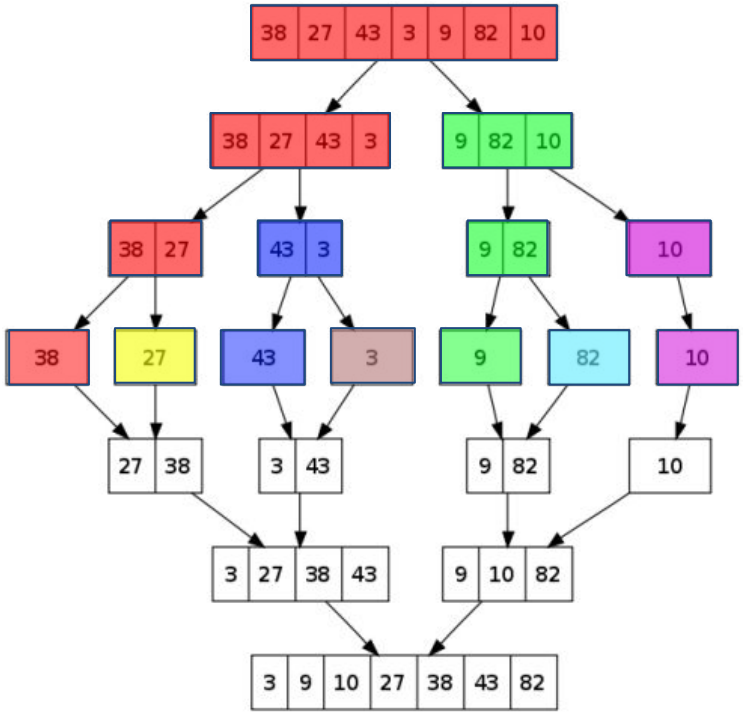


Parallel MPI Merge Sort

- Similar to GPU-based approach

- Recursively sub-divide problem across all available nodes
  - Send 2nd half to another processor
  - Worker processor starts traversing its leaf
  - When complete, sends sorted array to parent
  - Parent receives sorted array then merges with sorted 1st half.

- Once out of available processes, run serial mergesort
- When array sizes are small, use insertion sort instead

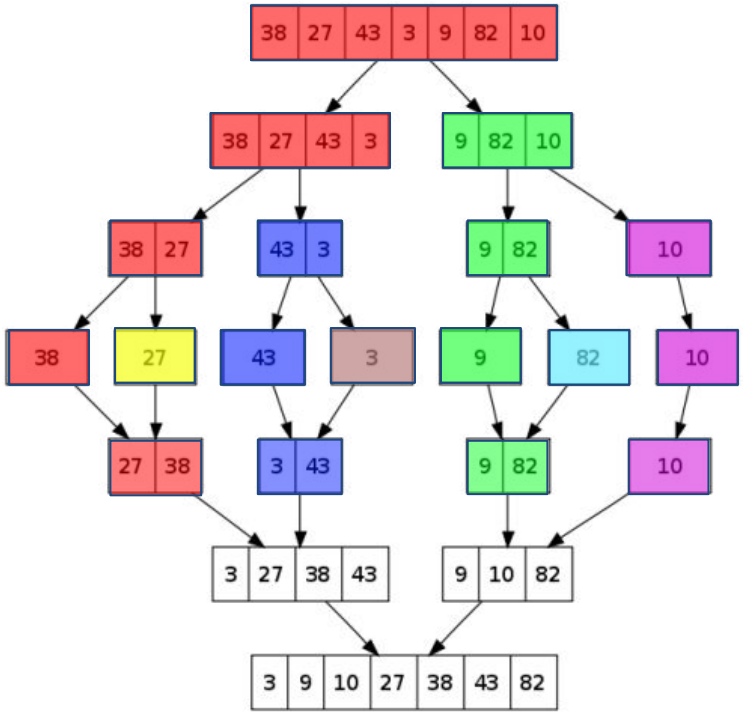


Parallel MPI Merge Sort

- Similar to GPU-based approach

- Recursively sub-divide problem across all available nodes
  - Send 2nd half to another processor
  - Worker processor starts traversing its leaf
  - When complete, sends sorted array to parent
  - Parent receives sorted array then merges with sorted 1st half.

- Once out of available processes, run serial mergesort
- When array sizes are small, use insertion sort instead

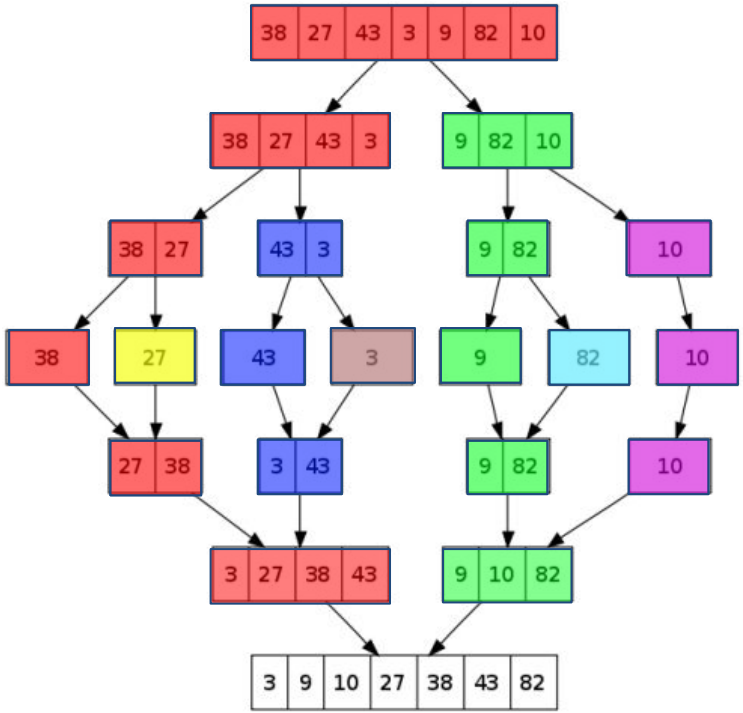


Parallel MPI Merge Sort

- Similar to GPU-based approach

- Recursively sub-divide problem across all available nodes
  - Send 2nd half to another processor
  - Worker processor starts traversing its leaf
  - When complete, sends sorted array to parent
  - Parent receives sorted array then merges with sorted 1st half.

- Once out of available processes, run serial mergesort
- When array sizes are small, use insertion sort instead

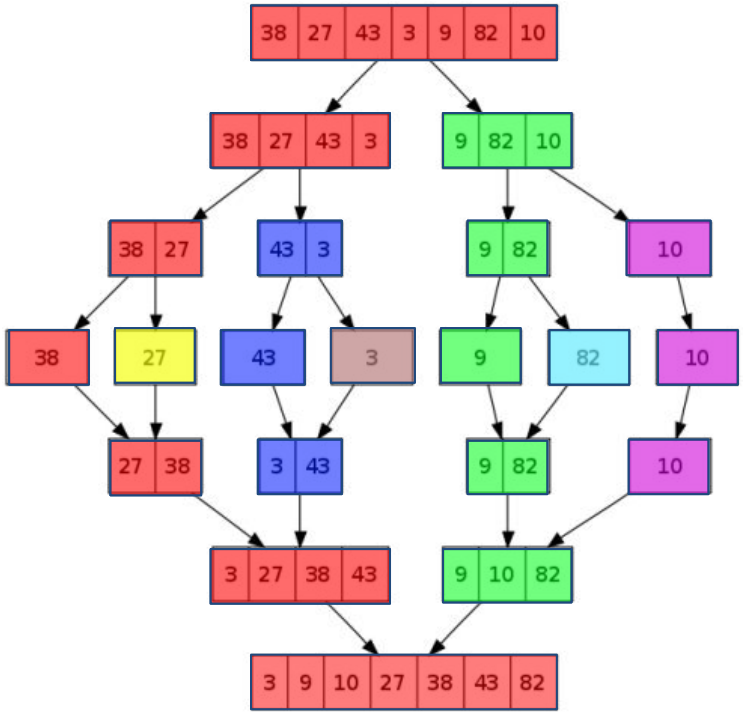


Parallel MPI Merge Sort

- Similar to GPU-based approach

- Recursively sub-divide problem across all available nodes
  - Send 2nd half to another processor
  - Worker processor starts traversing its leaf
  - When complete, sends sorted array to parent
  - Parent receives sorted array then merges with sorted 1st half.

- Once out of available processes, run serial mergesort
- When array sizes are small, use insertion sort instead

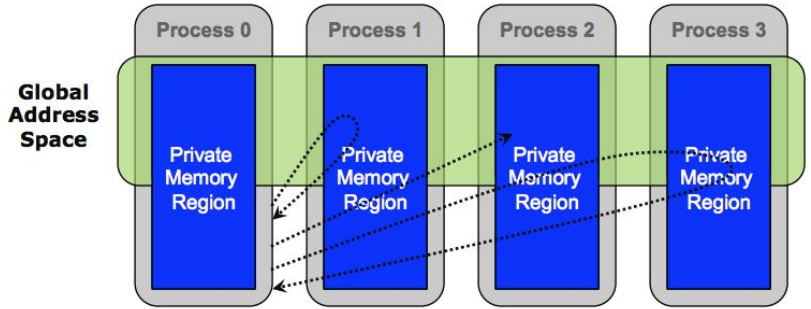


Parallel MPI Merge Sort - RMA

Remote Memory Access, or one-sided communication

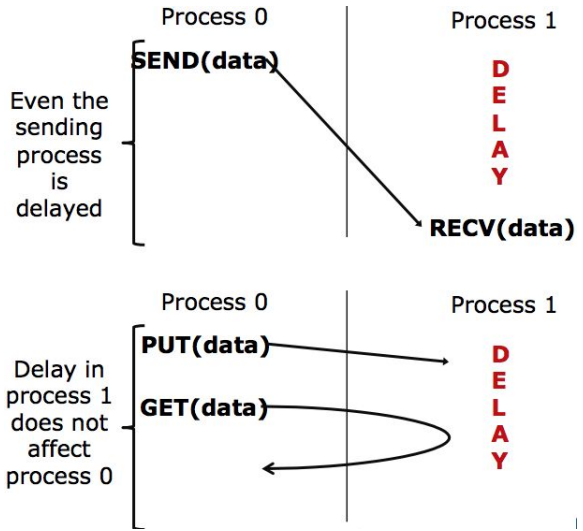
The basic idea of one-sided communication models is to decouple data movement with process synchronization

- Should be able to move data without requiring that the remote process synchronize
- Each process exposes a part of its memory to other processes
- Other processes can directly read from or write to this memory



Parallel MPI Merge Sort - RMA Advantages

- Can do multiple data transfers with a single synchronization operation
  - like BSP model
- Bypass tag matching
  - effectively precomputed as part of remote offset
- Some irregular communication patterns can be more economically expressed
- Can be significantly faster than send/receive on systems with hardware support for remote memory access, such as shared memory systems



Parallel Hybrid MPI / OpenMP Merge Sort

OpenMP - supports multi-platform shared memory multiprocessing programming

Hybrid Merge Sort - OpenMP is used for parallelism within a (multi-core) node while MPI is used for parallelism between nodes.

Advantages

- Reduces total number of MPI messages
- Lower latency communication (between threads)
- Load balancing
- Better fine grain parallelization

```
int main(int argc, char **argv)
{
    int a[100000];

    #pragma omp parallel for
    for (int i = 0; i < 100000; i++) {
        a[i] = 2 * i;
    }

    return 0;
}
```

Parallel MPI Merge Sort - Machines

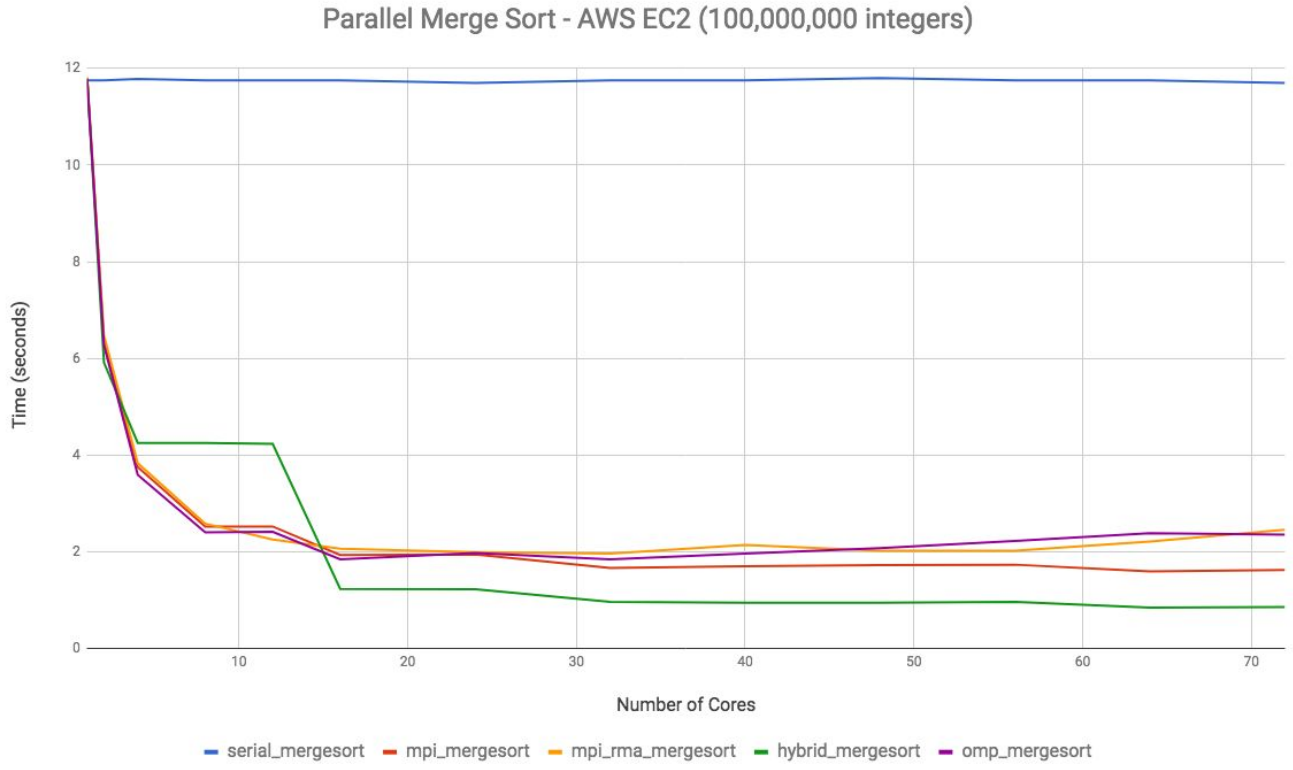
AWS EC2 - C5.18xlarge

- 1 instance
- Intel(R) Xeon(R) Platinum 8124M CPU @ 3.00GHz
  - 36 core (72 with HT)
  - 144 GB Memory

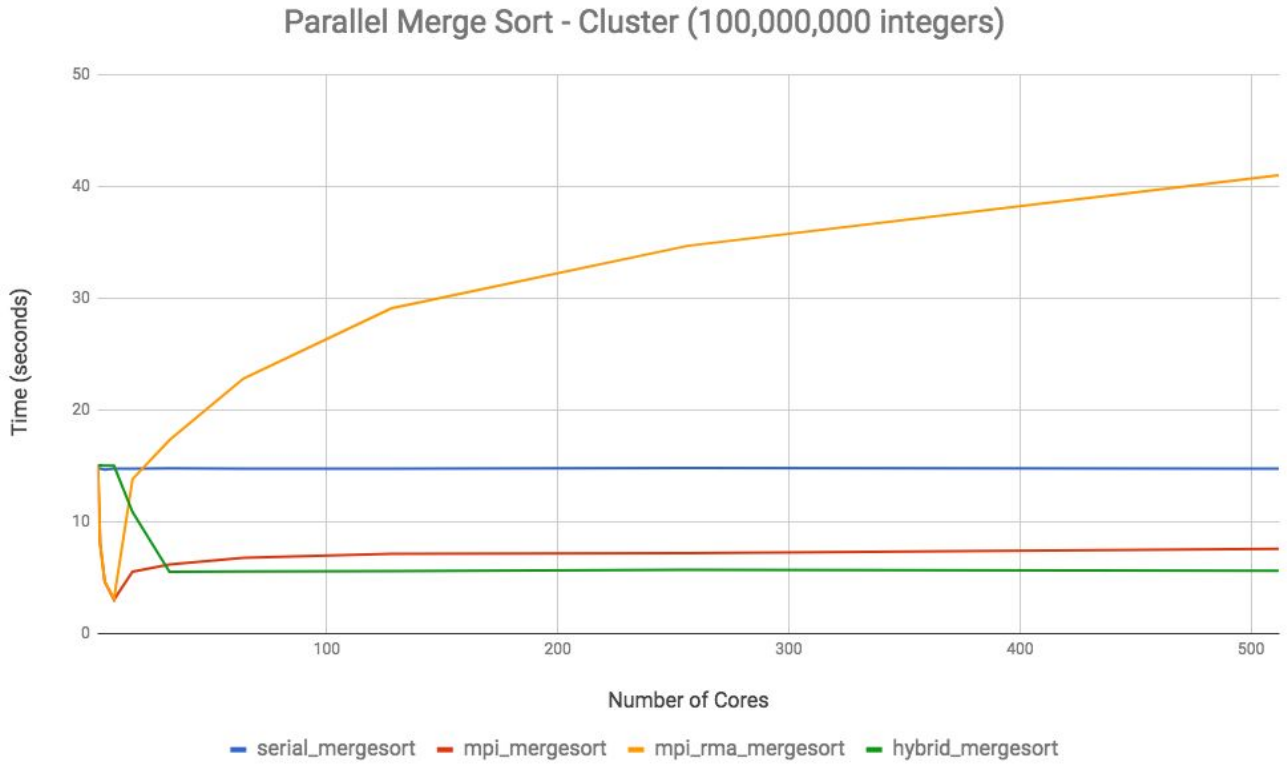
Cluster

- 60 Nodes, 720 cores total
- InfiniBand: Mellanox Technologies MT26428
- Intel(R) Xeon(R) CPU X5660 @ 2.80GHz
  - 6 cores (HT disabled)
  - 2 sockets per node
  - 24 GB Memory

Parallel MPI Merge Sort - AWS EC2



Parallel MPI Merge Sort - Cluster



Open Issues

- Challenging to program parallel algorithms
  - Requires expert knowledge (and even then it is hard)
  - OpenMP and OpenACC attempt to limit this burden
- CUDA is NVIDIA only
  - CUDA has many libraries (cuBLAS, cuDUNN, etc)
  - OpenCL, but not as well supported
  - Magma attempts to abstract away implementation for linear algebra
- Limited by memory
  - External memory sorting
  - Big Data

Discussion

Thank you for listening!

Questions

1. How much work Even-Odd sort does if there are *n* elements in a set?
2. Who invented first electromechanical punched card tabulator?
3. What are two conditions to watch out for when using MPI?

References

[1] D.E.Knuth, “The Art of Computer Programming”, Vol.3, pg.382

[2] S.G. Aki, “Parallel Sorting Algorithms”

[3] Algorithmist TM, http://www.algorithmist.com/index.php/Merge_sort, 19 May 2012

[4] Radenski, Atanas. "Shared memory, message passing, and hybrid merge sorts for standalone and clustered SMPs." (2011).

[5] Dagum, Leonardo, and Ramesh Menon. "OpenMP: an industry standard API for shared-memory programming." IEEE computational science and engineering 5, no. 1 (1998): 46-55.

[6] Gabriel, Edgar, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Vishal Sahay et al. "Open MPI: Goals, concept, and design of a next generation MPI implementation." In European Parallel Virtual Machine/Message Passing Interface Users’ Group Meeting, pp. 97-104. Springer, Berlin, Heidelberg, 2004.

[7] CUDA C/C++ Basics, “http://www.nvidia.com/docs/IO/116711/sc11-cuda-c-basics.pdf”, 2011