Ran ZHENG, Yuan-dong LIU, Hai JIN, 2020. Optimizing non-coalesced memory access for irregular applications with GPU computing.
Frontiers of Information Technology & Electronic Engineering, 21(9):1285-1301.
https://doi.org/10.1631/FITEE.1900262

Optimizing non-coalesced memory access for irregular applications with GPU computing

Front Inform Technol Electron Eng

Key words: General purpose graphics processing units; Memory coalescing; Non-coalesced memory access; Data reordering

Corresponding author: Ran ZHENG
E-mail: zhraner@hust.edu.cn
ORCID: https://orcid.org/0000-0002-3058-7581

Background

- GPUs can significantly accelerate many regular, data-parallel applications. However, the benefits of GPUs are not as useful for irregular applications.
- The execution time of kernels with irregular memory access consumes a large fraction in some irregular applications.

Table 1 Time proportions of irregular applications

Program	Time proportion		
	PCI-e	Irregular kernels	Others
CG	7%	85%	8%
SP	1%	94%	5%
MD	49%	51%	–

CG: conjugate gradient; SP: survey propagation; MD: molecular dynamics

Front Inform Technol Electron Eng

- Improving performance by optimizing the irregular memory access in irregular applications has great potential benefit.

Related work and challenges

Several solutions have been proposed to solve the irregularity problem.

- Hardware extensions: redesign the warp of a modern GPU
  - Highly dependent and limited by hardware
  - Expensive and difficult to make this a popular method
- Compiler techniques: reorder or relocate data before running the program
  - Optimize only the memory access pattern in compile time
  - Useful for static memory access but inapplicable to dynamic memory access
- Duplication algorithm: reordering data on CPUs
  - Inefficient, and high overhead for large-scale data
  - Extra overhead for transferring the reordered data

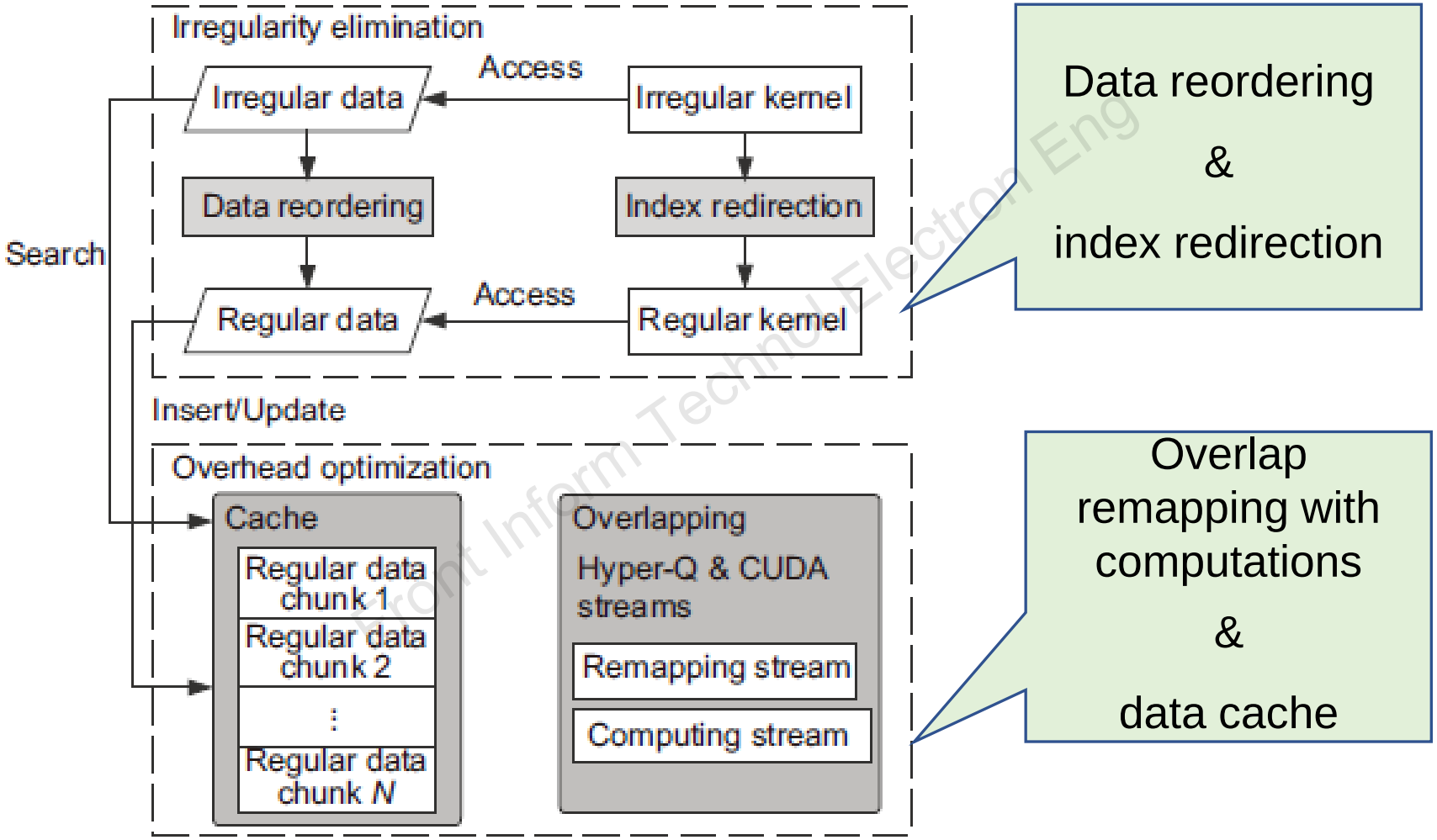
Front Inform Technol Electron Eng

Main idea

- Software solution to solve dynamic irregularities
  - Data reordering: divide data into multiple chunks and reorder chunk by chunk in accordance with the access patterns.
  - Index redirection: update and redirect the GPU kernel to access the reordered data.
- Cache data copy for reuse in the next iteration or other kernel
- Multiple CUDA streams and pipeline mechanisms to reduce the overhead of data reordering
  - Overlap the reordering and computations with Hyper-Q technology.

Front Inform Technol Electron Eng

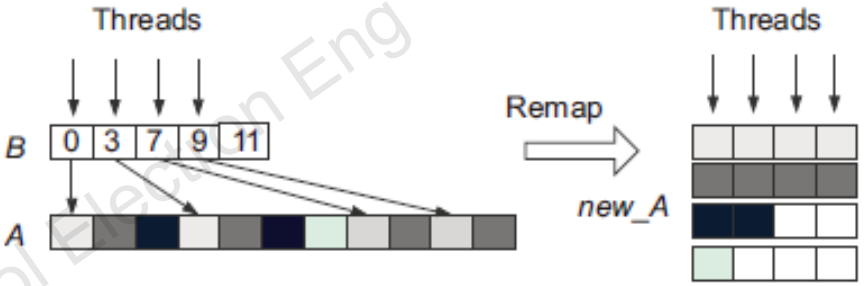
Scheme of eliminating irregular memory access

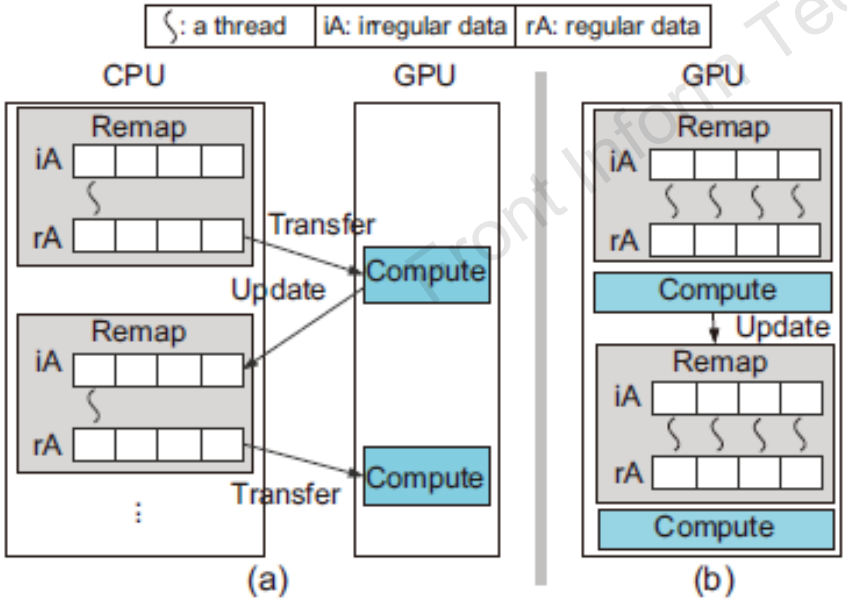


1. Dynamic data reordering

- Remove indirect memory access: extract elements in *A* indexed by *B* and fill these elements into the new memory region individually

$$new\_A[i + iter \cdot n] = \begin{cases} A[B[i] + iter], & 0 \le iter < B[i+1] - B[i], \\ trivial\ values, & \text{otherwise}, \end{cases}$$



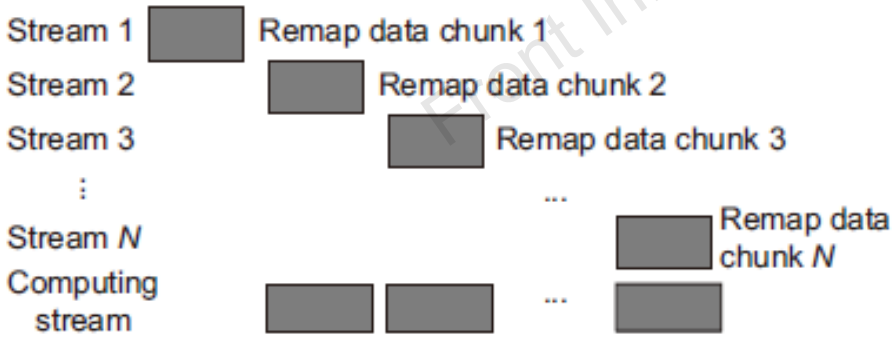


- Offload the reordering task from a CPU to a GPU
  - Launch a remapping kernel for each data chunk
  - Copy or fill one element into a new memory region in one kernel thread

Front Inform Technol Electron Eng

2. Multiple CUDA streams and pipeline

- Each remapping kernel is positioned at a different stream for each data chunk
- CPU checks whether a data chunk has been reordered in a polling mode
- With hyper-Q, the new feature of the Kepler architecture, the stream number, can be set up to 32 on Kepler GPUs.



Listing 4 Code snippet of overlapping remapping and computing kernels

```
//Initialize the number of streams
int streams_num = DATA_CHUNK;

//Initialize the size of the data chunk
int chunk_size = size / DATA_CHUNK;

//Create one computing stream
cudaStream_t comp_stream;
cudaStreamCreate(comp_stream);

//Create multiple reordering streams
cudaStream_t *streams = (cudaStream_t *)
malloc (streams_num*sizeof(cudaStream_t));
for (int i = 0; i < stream_num; i++)
        cudaStreamCreate(&streams[i]);

//Overlap remapping and computing kernels
for (int i =0; i < stream_num; i++) {
  //Reorder data chunks concurrently
  data_remap<<<grid,block,0,streams[i]>>>
        (original_data + i * chunk_size,
  new_data+i*chunk_size,index_array);

  if (check_Done(new_data,i)) {
  //Execute the computing stream
  // asynchronously
   data_comp<<<grid,block,0,comp_stream>>>
        (new_data+i*chunk_size);
  }
}
```

Front Inform Technol Electron Eng

3. Data cache

Software cache mechanism to record every remapping

- Record the mapping between the original and the corresponding reordered data
- Check whether the data have been reordered already
  - If found: return a pointer of new memory locations to avoid redundant data reordering
  - Otherwise: launch a remapping kernel to reorder this chunk of data, and insert a new entry filled by remapping results to the cache
- A valid field indicates whether the values of the data chunk are changed to avoid extra data copies and save global memory

Front Inform Technol Electron Eng

Benefits of data reordering on GPU

Reduction of memory transactions

Table 4 Total load memory transactions of CG with different input sizes

Input size	CUSPARSE	Proposed reorder	Reduction ratio
$259\,789^2$	12 586 844	10 436 777	1.2:1
$525\,825^2$	9 737 331	4 786 588	2.0:1
$715\,176^2$	10 929 912	5 189 755	2.1:1
$999\,999^2$	11 351 417	4 857 436	2.3:1

The input size is the size of a sparse matrix's row and column

Table 5 Total load memory transactions of SP with different input sizes

Input size	LonestarGPU	Proposed reorder	Reduction ratio
16 800-4000-3	1 959 774	1 709 631	1.2:1
42 000-10 000-3	5 028 018	4 386 694	1.2:1

The input size is the numbers of clauses-literals-literals per clause

Table 6 Total load memory transactions of MD with different input sizes

Input size	SHOC	Proposed reorder	Reduction ratio
12 288	7 097 802	3 584 748	2.0:1
24 567	14 204 070	7 173 772	2.0:1
36 864	21 306 534	10 760 875	2.0:1
73 728	42 613 954	21 522 198	2.0:1

The input size is the number of molecules

Data reordering varying from CPU to GPU

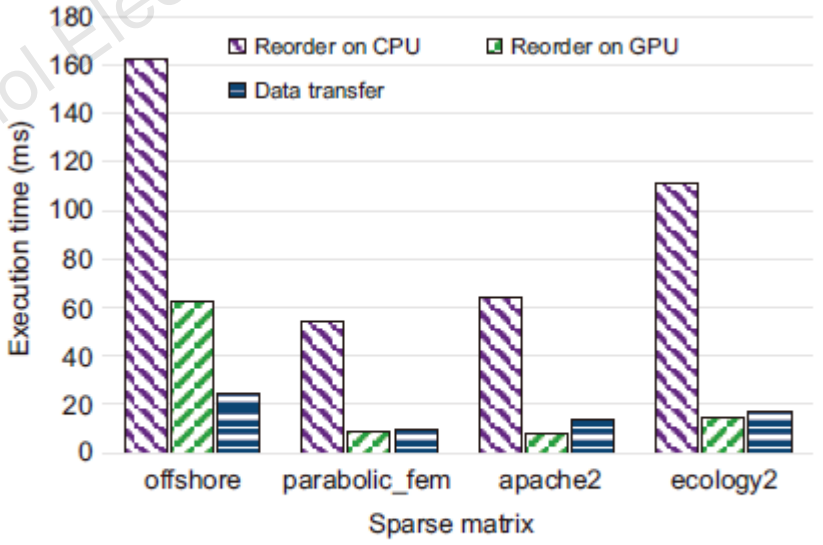


Fig. 6 Time comparison of data reordering on CPU and GPU

Front Inform Technol Electron Eng

Optimization with overlap and cache

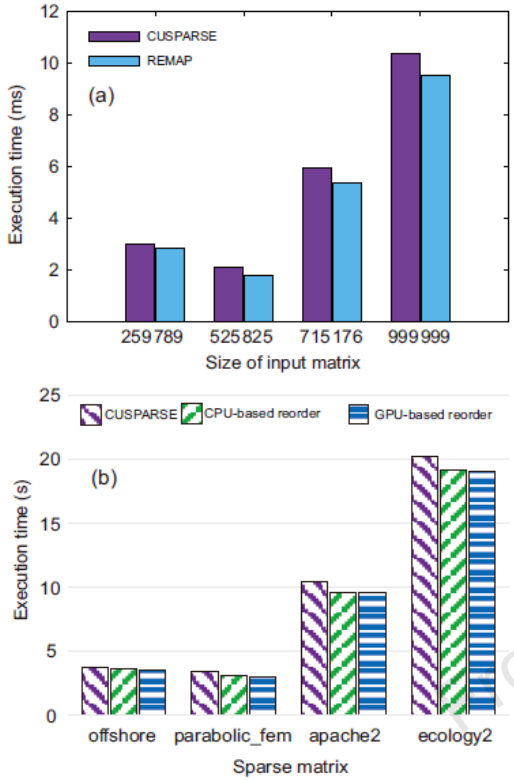


Fig. 7 Comparisons of the kernel and whole performance of the CG benchmark: (a) execution time of the *SpMV* kernel in each iteration; (b) execution time of the whole CG program

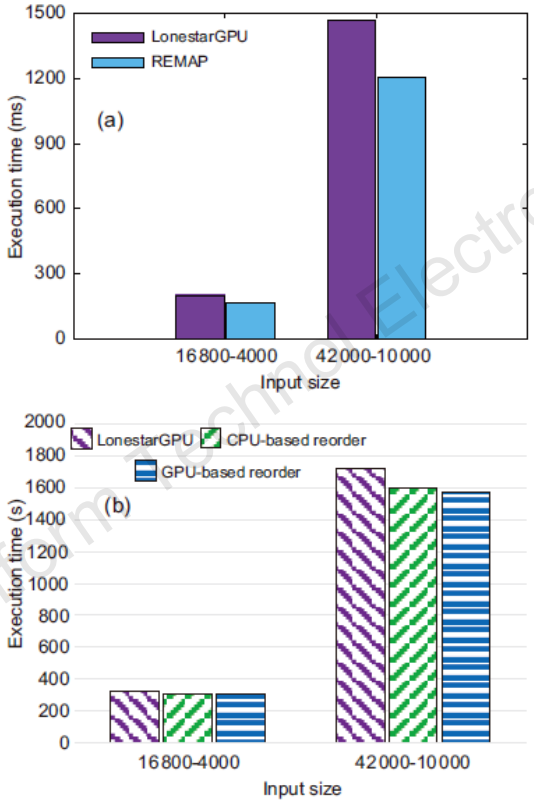


Fig. 8 Comparisons of the kernel and whole performance of the SP benchmark: (a) execution time of the *calc_pi_values* kernel in each iteration; (b) execution time of the whole SP program

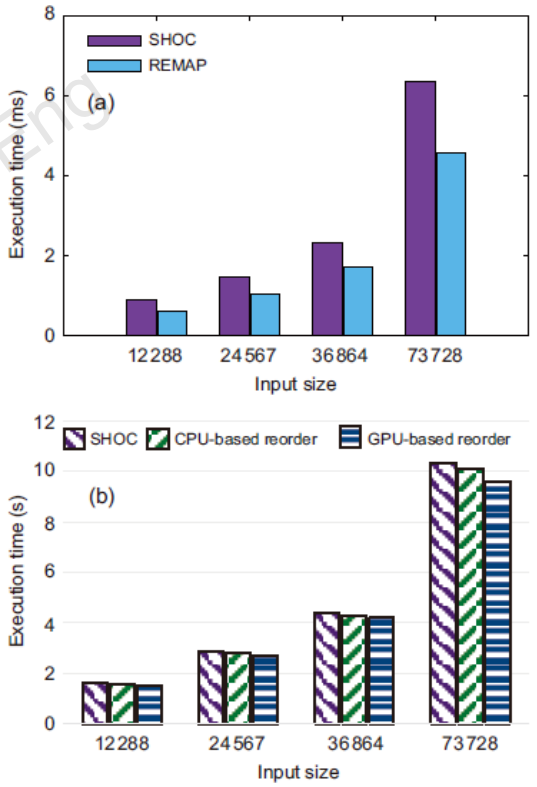


Fig. 9 Comparisons of the kernel and whole performance of the MD benchmark: (a) execution time of the *compute_lj_force* kernel in each iteration; (b) execution time of the whole MD program

Front Inform Technol Electron Eng

Conclusions

- The benefits of GPUs are minimal for irregular applications. To improve the performance of these irregular applications, a pure software solution to remove indirect memory access has been proposed.
- The volume of memory transactions can be reduced by 16.7%–50% compared with CUSPARSE-based benchmarks.
- The performance of irregular kernels can be improved by 9.64%–34.9% using an NVIDIA Tesla P4 GPU.

Front Inform Technol Electron Eng

Ran ZHENG received her MS and PhD degrees from Huazhong University of Science and Technology (HUST), China in 2002 and 2006, respectively. She is currently an Associate Professor of Computer Science and Engineering at HUST. Her research interests include distributed computing, cloud computing, high-performance computing, and their applications.

Hai JIN is a Cheung Kung Scholars Chair Professor of Computer Science and Engineering at Huazhong University of Science and Technology (HUST) in China. He received his PhD degree in Computer Engineering from HUST in 1994. In 1996, he was awarded a German Academic Exchange Service fellowship to visit the Technical University of Chemnitz in Germany. He worked at the University of Hong Kong between 1998 and 2000, and was a visiting scholar at the University of Southern California between 1999 and 2000. He was supported by the National Science Fund for Distinguished Young Scholars in 2001. He is the Chief Scientist of ChinaGrid, the largest grid computing project in China, and the Chief Scientist of National 973 Basic Research Program Project of Virtualization Technology of Computing System, and Cloud Security. He is an editorial board member of *Frontiers of Information Technology & Electronic Engineering*. His research interests include computer architecture, virtualization technology, cluster computing and cloud computing, peer-to-peer computing, network storage, and network security.

Front Inform Technol Electron Eng