Using Integrated Processor Graphics to Accelerate Concurrent Data and Index Structures

Joel Fuentes
University of California, Irvine

SC18 Dallas, TX hpc inspires.

Abstract

We proposed different data and index structure algorithms that can benefit from the iGPU architecture and programming model. Experimental results show speedups of up to 4x on concurrent data structures and 11x on index structures, when comparing with state-of-the-art CPU implementations. Energy savings of up to 300% are also obtained when running these algorithms on iGPU.

Dissertation goals

1. Demonstrate that Intel's iGPU provides support and a set of instructions to implement concurrent data and index structures efficiently.
2. Find concurrent data structures that can be ported to C for Media for iGPU.
3. Propose a general algorithm to construct index structures and execute queries on iGPU.
4. Achieve better performance and energy savings compared to CPU.

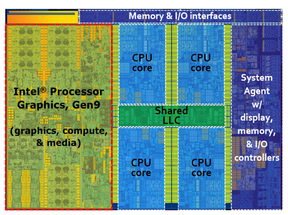
Acknowledgments

We are grateful to Intel Corporation and the CM Compiler team for providing us the computing resources needed for this research.

Intel iGPU and C for Media

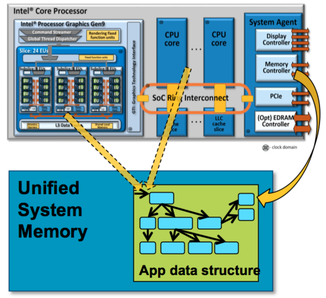
GenX Architecture

GenX is the architecture for Intel integrated processor graphics.



- Integration of multiple CPU cores and Intel processor graphics (called GenX) on the same die.
- Bi-directional ring interconnect that has a 32-byte wide bus
- Last Level Cache (LLC) and embedded EDRAM exclusively for iGPU.

- Shared Virtual Memory between CPU cores and iGPU.
- Execution unit (EU) is the foundational building block of GenX.
  - Each EU has 7 threads.
  - Each thread has 128 SIMD8 32-bit registers (GRF).
  - Each EU has 2 FPUs (SIMD floating-point units), one branch unit and one send unit



C for Media (CM)

- High-level programming environment based on C/C++.
- Language designed to efficiently leverage SIMD capability of the iGPU
- Two primary components:
  1. The compiler for the CM language
  2. CM runtime, which provides an API for C/C++ programs

Cm operations

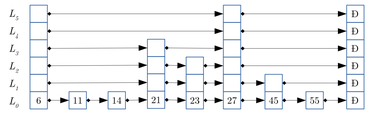
- SIMD atomic operation
- Select and Merge
- Format
- Boolean reductions

```
1
2
3  template <typename T> inline vector<T, 16> PrefixSumIn(vector<T, 16> in) {
4    vector<T, 32> d = 0;
5    d.select<16, 1>(16) = in;
6
7    d.select<16, 1>(16) = d.select<16, 1>(16) + d.select<16, 1>(15);
8    d.select<16, 1>(16) = d.select<16, 1>(16) + d.select<16, 1>(14);
9    d.select<16, 1>(16) = d.select<16, 1>(16) + d.select<16, 1>(12);
10   d.select<16, 1>(16) = d.select<16, 1>(16) + d.select<16, 1>(8);
11
12   return d.select<16, 1>(16);
13 }
14
```

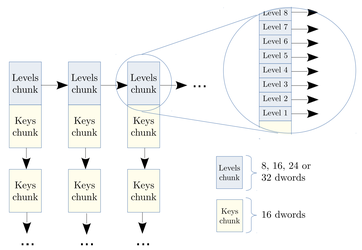
Concurrent Data Structures

Lock-free Skiplist

Probabilistic data structure that is built upon the general idea of a linked list It provides well performance for concurrent operations, while still providing $O(logn)$ complexity for all its operations.

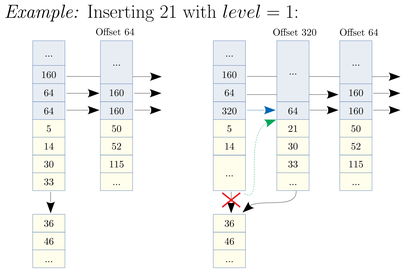


Skiplist design (CMSL for search, insert and delete):



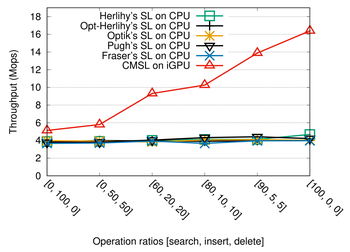
- Chunks of links and keys are used instead of nodes
  - GenX supports SIMD16. Update operations are performed on entire chunks.
- Chunk of links (blue) and keys (yellow):
  - Chunk of links stores offsets of next lists. Chunk of keys stores sorted keys
  - 1 dword is reserved for a link to next chunk within the same list

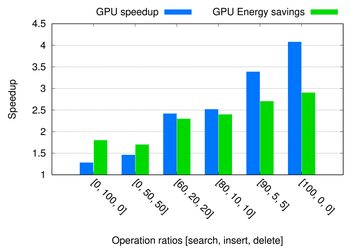
Example: Inserting 21 with $level = 1$:



1. Search list and position of 21
2. Create new list. Steal $keys > 21$ from previous list, including attached chunk
3. Perform SIMD16 atomic operation on previous list to remove $keys > 21$
4. Perform atomic operations on upper links to update offset (320) while $level \leq 1$

Experimental results: Search, insert, delete operations on a Skiplist with 1M keys. Compared with state-of-the-art Skiplists: Herlihy et al., Guerraoui et al., Pugh and Fraser. Experiments carried out on a Intel(R) Core i7-8670HQ CPU, with a total of 4 physical cores, 8 logical processors and Intel(R) Iris Pro Graphics 580 (GT4e and Gen9 microarchitecture).





Comparison with the M&C's Skiplist [3] for discrete GPU. Experiments carried on a Nvidia GTX 970. Relative efficiency is calculated by $efficiency = 1/(time * peak\_GFLOPS)$. Intel iGPU has a peak GFlops of 1,152. Nvidia GTX 970 has a peak GFlops of 3,920.

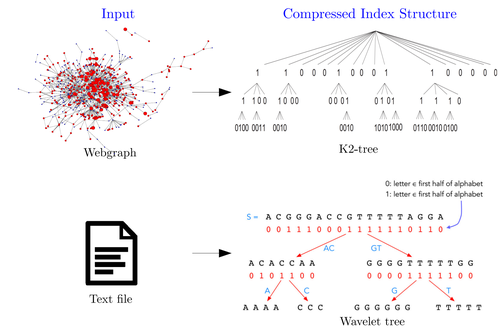
Ops. [search, insert, delete]	CMSL (ms)	M&C (ms)	Relative efficiency
[0, 100, 0]	244	110	1.5
[0, 50, 50]	212	110	1.6
[60, 20, 20]	141	69	1.75
[80, 10, 10]	93	48	1.8
[90, 5, 5]	72	43	2.0
[100, 0, 0]	61	44	2.5

Experimental Results

The proposed Skiplist performs up to **4.3x faster** running on the iGPU compared with other Skiplists running on CPU. The best performance is achieved with low contention of operations: mostly search operations. The use of iGPU also produces **300% of energy savings**, compared with same execution on CPU cores on the same die.

Index Structures

Index structures are widely used by search engines, databases, graphics applications, etc. Compressed index structures reduce space of regular indices while allowing efficient navigation in compressed form. As such, they allow running main-memory algorithms on much larger structures. However, their construction is time-consuming. We studied the following index structures to propose their efficient implementation on iGPU: K^2-tree, Wavelet tree and Suffix tree.

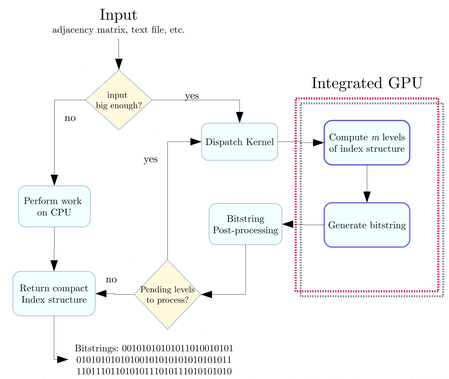


- K^2-tree:
  - Stores a webgraph in compressed space.
  - Input (adjacency matrix) is processed by levels bottom-up.
  - Final compressed representation is bitstrings T and L.
- Wavelet tree:
  - Stores strings/text in compressed space.
  - Input (text) is processed by levels top-down.

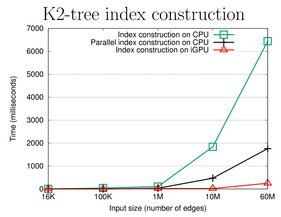
Queries on compressed index structures are performed using $rank_q$ and $select_q$ operations defined on bitstrings.

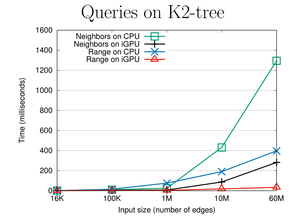
General algorithm to construct compressed index structures:

1. iGPU kernels are dispatched only if the input is big enough. i.e. more than 1M edges for K^2-tree.
2. Most of index structures represent trees and they are processed level by level (bottom-up or top-down).
3. In some index structures, e.g. K^2-tree, CPU post-processing is needed after every kernel dispatch.



Experimental results: index construction and queries on iGPU compared with SDSL-lite on CPU.





Experimental Results

The construction of the index structure K^2-tree is **7 times faster** on iGPU. **Speed-up on queries is up to 11x**.

Want to learn more?

Scan me

References

[1] Herlihy, Maurice and Lev, Yossi and Luchangco, Victor and Shavit, Nir. *A simple optimistic skiplist algorithm.* 2007.

[2] Pugh, William. *Skip lists: a probabilistic alternative to balanced trees.*1990.

[3] Misra, Prabhakar and Chaudhuri, Mainak. *Performance evaluation of concurrent lock-free data structures on GPUs.* 2012. =