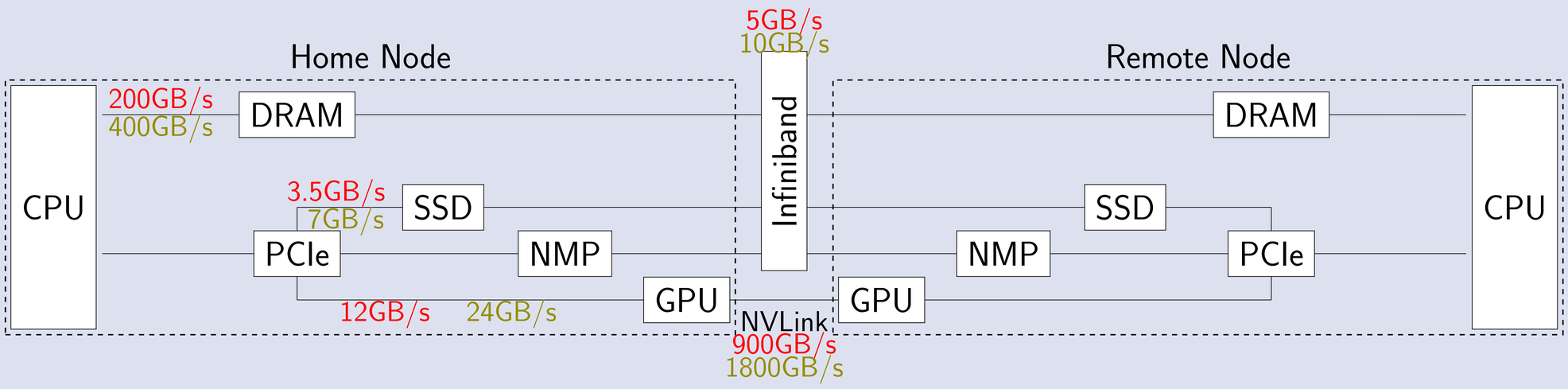
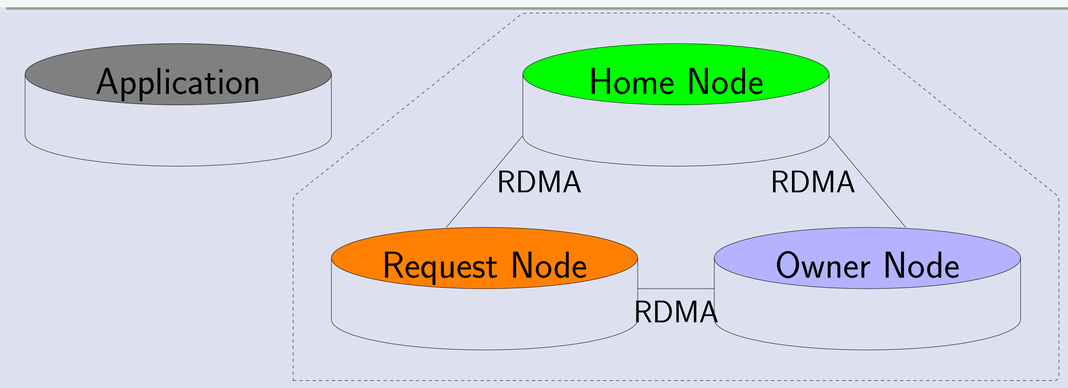
Buffers with Benefits

Elastic Memory Hierarchies for Memory-Intensive Applications

Compression leads to a higher effective bandwidth (factor two, green).

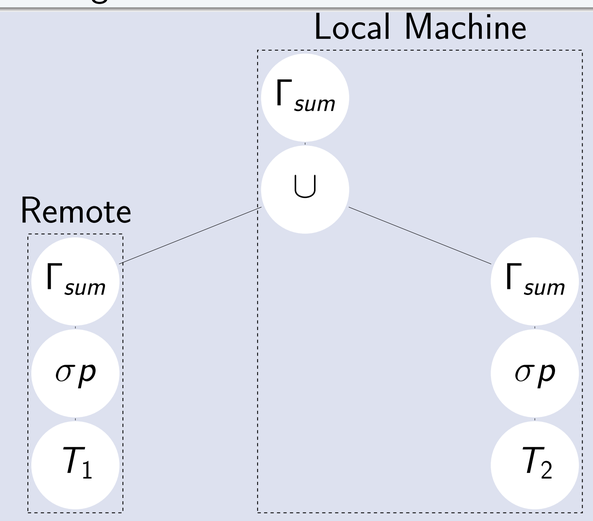


Cache-coherence protocol over RDMA

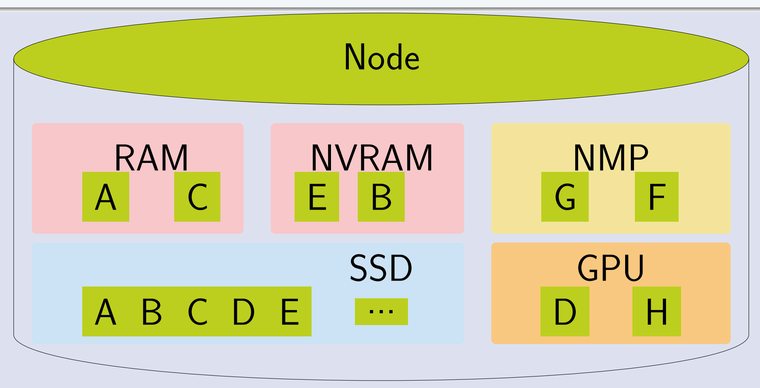


Distributed query processing

When data is partitioned horizontally ($T = T_1 \cup T_2$), remote nodes can perform computations such as table scans, aggregations and selections locally before transferring the data.

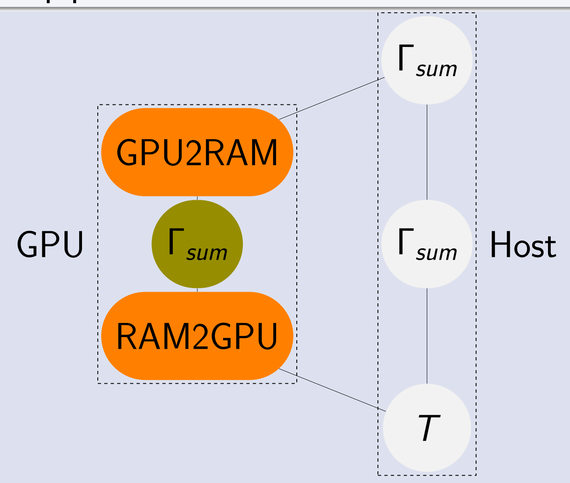


Multi-tier buffer management extended to store data on GPU, NMP or NVRAM where computation takes place



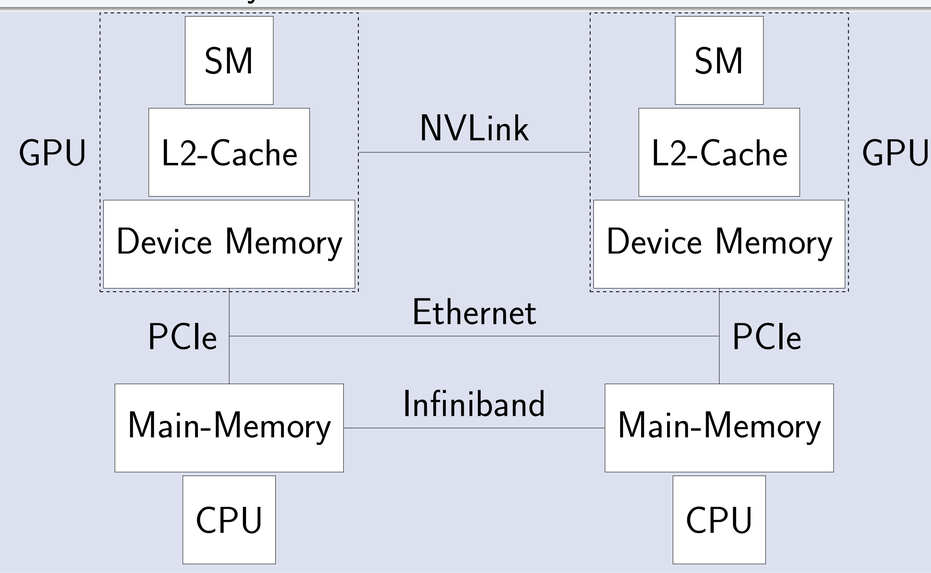
Query plan out of table scan and an aggregation

Parts can be off-loaded to GPUs. However, memory transfer needs to take place before performing the aggregation on the GPU and once before finalizing the query on the host CPU. With a unified memory, memory transfers can be omitted. However, changing a device for processing means breaking the pipeline.



NVLink

Traditionally, GPUs are connected via PCIe for memory transfer between device memory and main-memory. RDMA technology such as Infiniband allows direct memory access and bypassing the CPU. Similarly, NVLink connect GPUs directly.



Bandwidth of inter-GPU communication for selected GPUs under either PCIe or NVLink.

Sources: https://en.wikipedia.org/wiki/NVLink
https://www.nvidia.com/de-de/design-visualization/nvlink-bridges/

GPU Model	Interconnect	Transmissions	Lanes	Data rate (out + in)
V100	PCIe 3.0	8 GT/s	16 + 16	16 + 16 GB/s
V100	NVLink 2.0	25 GT/s	8 + 8	150 + 150 GB/s
GeForce RTX 2080 Ti	PCIe 3.0	8 GT/s	16 + 16	16 + 16 GB/s
GeForce RTX 2080 Ti	NVLink 3.0	25 GT/s	8 + 8	50 + 50 GB/s
Ampere A100	PCIe 4.0	16 GT/s	4 + 4	32 + 32 GB/s
Ampere A100	NVLink 3.0	50 GT/s	4 + 4	300 + 300 GB/s

Code-Generation for GPU DBMS

```
1  class Operator {
2   virtual void produce(Codegen *codegen)=0;
3   virtual void consume(Codegen *codegen) {}
4  };
5  // wrapper class for unary operators
6  class UnaryOperator : public Operator {
7   protected:
8   std::unique_ptr<Operator> child=nullptr;
9   public:
10  // produce:  ask child for tuples
11  void produce(Codegen *codegen) override{
12   if (child) child->produce(codegen);
13  }
14 };
```

Listing 1: *Interface within the namespace `codegen` with abstract operator classes.*

```
1  class GPUTableScan : public UnaryOperator  {
2   std::string size, table;
3   public:
4   // constructor takes size, use id of each gpu thread
5   GPUTableScan(std::string size, std::string
        table) : size(size), table(table) {}
6   // produce starts the pipeline by tablescan
7   void produce(Codegen *codegen) override{
8    codegen->add2header("unsigned int n");
9    codegen->add2header(table + "_*d_" + table);
10   codegen->add("auto i=blockIdx.x*blockDim.x+
         threadIdx.x;\nif(i<"+size+"){\n");
11   if (parent) parent->consume(codegen);
12   codegen->add("} //end GPUTableScan\n");
13  }};
```

Listing 2: *Class for GPU Table Scan.*

```
create table products(p_id int primary key,
    category int, price int);
create table sellers(s_id int primary key,
    region int);
create table sales(product_id int,seller_id
    int);
select category, sum(price)
from sales, sellers, products
where product_id = p_id and
      seller_id=s_id and region=0
group by category
```

Listing 3: *Exemplary query.*

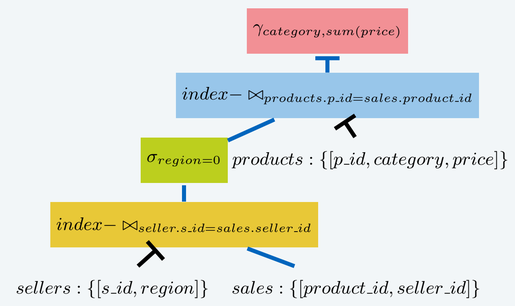


Figure 6: *Query plan to Listing 3 composed of an aggregation (red), two index-joins (blue, yellow) and a selection (green).*

```
1  extern "C" __global__ void kernel(
2   Sale *begin, unsigned int n, Product *d_products,
3   Seller *d_sellers, int capacity,
4   HTEntry *device_entries, unsigned int unused_key) {
5   auto i = blockIdx.x * blockDim.x + threadIdx.x;
6   if (i < n) {
7    auto &seller = d_sellers[begin[i].seller_id];
8    if (seller.seller_region==0u){
8     auto &product = d_products[begin[i].product_id];
9     int key = product.product_category;
10    int value = product.product_price;
11    int slot = get_hash(key) & (capacity - 1);
12    while (true) {
13     int old = atomicCAS(&device_entries[slot].key, unused_key, key);
14     if (old == unused_key  old == key) {
15      atomicAdd(&device_entries[slot].value, value);
16      break;
17     }
18     slot = (slot + 1) & (capacity - 1);
19  }}}}
```

Listing 4: *Generated GPU kernel using index-joins (blue, yellow), a selection (green) and the hash-table for aggregation (red).*

Contact

Maximilian E. Schüle, PI; Maha M H Alwahibi, RA
maximilian.schuele@uni-bamberg.de
maha-m-h.alwahibi@stud.uni-bamberg.de

Funded by DFG Deutsche Forschungsgemeinschaft German Research Foundation
SPP 2377