**Content Defined Chunking**

|  |  |
| --- | --- |
| **Axis** | Max chunk size |
| **Challenge** | Tradeoff between varying chunk and window sizes and throughput of CDC |
| **Opportunity** | The smaller the max chunk size, the les number of bits that need to be used in the rolling hash computation. The time required per hash computation is independent of the rolling window size, since we use a rolling hash function. |
| **Continuum** | Reducing max chunk size allows the creation of more chunks and increases the number of SHA and LZW computations |
| **Equation for Benefit** | *Chunk Size = c1 \* (1 / SHA comparisons)*  *Chunk Size = c2 \* (1 / LZW codes)*  *Chunk Size = c3 \* (# of chunks)* |

|  |  |
| --- | --- |
| **Axis** | Multiple CDC compute units |
| **Challenge** | Improving throughputs of chunks created from input stream. Ideally linear increase in throughput as we add more CDC units. |
| **Opportunity** | The chunks generated from the split stages can only be utilized by further stages of the pipeline in a FIFO manner of the chunks |
| **Continuum** | The chunkable data stream would reduce thus creating more overhead in LZW compression. |
| **Equation for Benefit** | *Total throughput = (# of CDC Modules) \* (Throughput of Individual CDC modules)* |

|  |  |
| --- | --- |
| **Axis** | Using vectors to reduce Initialization Interval of the pipeline |
| **Challenge** | Improving throughput by leveraging data level parallelism |
| **Opportunity** | Data inside the CDC is processed in a bitwise manner, and there is no data independent operation. The start of one chunk depends on the end of the previous chunk. |

|  |  |
| --- | --- |
| **Axis** | Optimizing memory accesses |
| **Challenge** | Alleviating duplicate DRAM accesses which can bottleneck the FPGA. Use caches to prevent DRAM access penalties. |
| **Opportunity** | If the same computation is repeated multiple times it can be efficient to cache the result of the computation. As an example, the SHA computation requires arithmetic operations which can be reused across passes. |
| **Continuum** | Use local registers for intermediate storage of computations. |
| **Equation for Benefit** | *Eliminate clock cycles associated with redundant memory accesses.* |

|  |  |
| --- | --- |
| **Axis** | Splitting up the computation between FPGA and CPU resources |
| **Challenge** | Splitting up the CDC functionality between the FPGA and CPU resources |
| **Opportunity** | Exploiting data level parallelism using the CPU vector engine. CPU computations can benefit from caches and higher clock speeds, whereas FPGA computations offer application specific hardware based on their configuration. |
| **Continuum** | Which stages of the CDC computation will get split between CPU cores, and FPGA hardware. |

**SHA Hash**

|  |  |
| --- | --- |
| **Axis** |  |
| **Challenge** |  |
| **Opportunity** |  |
| **Continuum** |  |
| **Equation for Resources** |  |

|  |  |
| --- | --- |
| **Axis** | Throughput; chunk size; number of chunks; input size; II; memory |
| **Challenge** | Improving the throughput; optimizing the II |
| **Opportunity** | Parallelism with LZW; pipelining; multiple-threads parallelism |
| **Continuum** | Range from 1 to *number\_of\_chunks* hashes |
| **Equation for Benefit** | *N* threads parallelism: *T(N)=T(1)/N;*  Parallelism with LZW: *T=max(T\_SHA, T\_LZW)* |
| **Equation for Resources** | Resource(*R*)=*R\*single\_threshold\_resource* |

Deduplication:

|  |  |
| --- | --- |
| **Axis** | Throughput; input size; memory; chunk size; hashmap; hashes |
| **Challenge** | Improving the throughput; dependency on SHA256; |
| **Opportunity** | Parallelism |
| **Continuum** | Range from 1 to *number\_of\_chunks* hashes |
| **Equation for Benefit** | Running *N* storing in parallel: *T(N)=T(1)/N* |
| **Equation for Resources** | Resources(*R*)= *R\*single\_storage\_resource* |

LZW:

|  |  |
| --- | --- |
| **Axis** | Throughput; input size; memory; pointers; encoding; II |
| **Challenge** | Improving the throughput; optimizing the II |
| **Opportunity** | Parallelism with SHA; apply parallelism in LZW; pipelining |
| **Continuum** | Range from 1 to input\_size comparisons and memory lookup |
| **Equation for Benefit** | Running *N* comparison and lookup in parallel: *T(N)=T(1)/N* |
| **Equation for Resources** | Resources(*R*)= *R\*(single\_comparison\_resource+single\_lookup\_resource)* |

Communication/integration:

|  |  |
| --- | --- |
| **Axis** | Data rate among operations; input size; memory; hashmap; chunk size; pipeline |
| **Challenge** | Improving the data rate; data dependency |
| **Opportunity** | SHA and LZW may run in parallel |
| **Continuum** | All the data transmission processes |
| **Equation for Benefit** | Parallelism with LZW: *T=max(T\_SHA, T\_LZW)* |
| **Equation for Resources** | Resource = *Resource\_SHA*+*Resource\_LZW* |