

PBP as Efficient Bit-Serial SIMD



Parallel bit pattern (PBP) computing is a quantum-inspired computation model, but it was not created to replace quantum computing: *the goal is to reduce power per computation by orders of magnitude.*

LCPC17: How Low Can You Go?

Our 2017 paper at Languages and Compilers for Parallel Computing, DOI 10.1007/978-3-030-35225-7_8, observed that the best way to reduce power/computation is to eliminate unnecessary gate-level operations:

- Work only on active bits (bit-serial), not words
- Aggressively optimize computations at the gate-level
- Leverage entangled superposition

As our LCPC22 paper, *Wordless Integer and Floating-Point Computing*, describes we now have a C++ library that accomplishes all three of those goals. It not only implements quantum-like **pbit (pattern bit)**, but also **pint (pattern integer)** and **pfloat (pattern float)** with runtime variable precision and compiler-like symbolic optimization at the pbit and lower levels. The interesting thing is that **pbit** entangled superpositions are not used to implement quantum-like computation, but to dramatically improve the efficiency of bit-serial SIMD execution.

Bit-Serial SIMD Execution

Bit-serial processing basically means that multi-bit values are evaluated one bit position at a time. Consider:

```
int a, b, c; c = a + b;
```

Using 32-bit words, a carry lookahead addition would require **~645 gate actions** to produce one 32-bit result every clock cycle. Using ripple carry would only take **~153 gate actions**, and could be done bit serially in 32 faster clock cycles. The throughput can then be multiplied by having many bit-serial SIMD processing elements; this was done in early supercomputers including **MPP** and **CM1/CM2**.

We can do much better with a little runtime symbolic optimization. If the current values of **a** and **b** each fit in just 4 bits, we only need **17 gate actions** or four clock cycles. If **b** happens to be 1, instead of a 32-bit adder, a 4-bit incrementer with just **7 gate actions** suffices! Our PBP C++ library tracks precision of both **pint** and **pfloat** data, also applying symbolic optimization at the **pbit** level to avoid unnecessary gate-level operations.

Leveraging PBP Entangled Superposition

An E -way entangled **pbit** is logically equivalent to an array of 2^E bit values, one bit in each of 2^E fully addressable **entanglement channels** – each of which can be treated as a virtual SIMD processing element. For example, consider a **pint** that holds the PE number, from 0 to 31, in each of 32 PEs ($E=5$). With PE0 in the rightmost entanglement channel, this would look like:

```
10101010 10101010 10101010 10101010
11001100 11001100 11001100 11001100
11110000 11110000 11110000 11110000
11111111 00000000 11111111 00000000
11111111 11111111 00000000 00000000
```

The grouping into **chunks** of 8 channels each isn't just to increase readability here; PBP fragments **pbit** values into chunks of 2^K bits. Only a single copy of each unique **chunk pattern** is stored and a pbit's value is actually stored as a **regular expression** treating each chunk as a **symbol**. Thus, the example would really be:

```
chunk (2) chunk (2) chunk (2) chunk (2)
chunk (3) chunk (3) chunk (3) chunk (3)
chunk (4) chunk (4) chunk (4) chunk (4)
chunk (1) chunk (0) chunk (1) chunk (0)
chunk (1) chunk (1) chunk (0) chunk (0)
```

and would use only $5*8=40$ bits of storage, not 160. Symbolic analysis eliminates many chunk operations: for example, **chunk (1) & chunk (42)** is **chunk (42)** without examining any bits, and any chunk operation that has been done before on any PEs is available to all, hence is never repeated. This is a huge reduction in gate operations needed. A PBP chunk behaves like a generalization of a GPU **warp**, allowing computation to be skipped under far more circumstances than just when all component PEs are disabled. Of course, chunks are typically $K \geq 8$, not $K=3$, and up to 4G PEs are supported.



In Our SC24 Exhibit (booth #4525)

The big new piece of hardware in our research exhibit is **KES**, which uses LEDs under each of six "Q-bits" to show the PBP chunked bit vectors. Each **row** of LEDs in KES corresponds to one of 64 bit-serial PEs within a SIMD engine, which comprises eight $K=3$ SIMD chunk engines. Only chunk computations producing LEDs lit **red** require bit-level work, so the dramatic reduction in total gate-level actions needed is obvious. These chunk computations could be scheduled using a pool of compute resources containing any number of $K=3$ SIMD chunk engines.

The little new piece of hardware is **PBPx4**, which uses an **aggregate function network** to coordinate four $K=10$ SIMD chunk processors implemented using FPGAs. When fully operational, we expect PBPx4 to prove our claims of orders of magnitude power reduction using PBP.

```
@techreport{sc24pint,
author={Dietz, Henry},
title={{PBP as Efficient Bit-Serial SIMD}},
institution={{University of Kentucky}},
url={{http://aggregate.org/WHITE/sc24pint.pdf}},
month={Nov}, year=2024}
```

