Vectorising and distributing NTTs to count Goldbach partitions on Arm-based supercomputers

Ricardo Jesus[1] Tomás Oliveira e Silva[2] Michèle Weiland[1]

[1]EPCC, The University of Edinburgh

[2]IEETA/DETI, Universidade de Aveiro

May 2021

Contact: `rjj@ed.ac.uk`

- Introduction
- Vectorising modular arithmetic loops with SVE
- Distributing and parallelising NTTs
- Preliminary results on counting Goldbach partitions
- Conclusions

- **Problem: count Goldbach partitions to large limits**
  - Evaluate for all even n below cutoff point

$$R(n) = \#\left\{(p,q) : n = p + q \wedge p, q \text{ prime}\right\} \tag{1}$$

  - Can be done via a polynomial product $\Rightarrow$ requires exact convolution
- **Number-theoretic transforms** (NTTs) enable **fast exact convolutions**
  - "DFTs with modular arithmetic"
  - Used extensively for bignum and polynomial arithmetic
  - Because we want to achieve very large limits, we need a distributed implementation
- NTTs rely on **modular arithmetic**
  - Tends to hinder vectorisation (lack of suitable instructions)
  - SVE supports the required operations

Modular arithmetic

- Elementary operations: addition, subtraction, and multiplication

$$z_\pm = (x \pm y) \bmod N; \quad z_\times = (xy) \bmod N$$

- Additions and subtractions are trivial, **multiplications are harder**
- The Montgomery method is a well-known algorithm for doing fast modular multiplication
  - For $R > N$ and $\gcd(R, N) = 1$:

$$\text{Let } \tilde{x} = (xR) \bmod N,\ N' = \left(-N^{-1}\right) \bmod R, \text{ and} \tag{2}$$

$$\text{REDC}(T) = \left[T + N\left((TN') \bmod R\right)\right]/R \bmod N^\dagger. \text{ Then} \tag{3}$$

$$\boxed{\widetilde{xy} = \text{REDC}(\tilde{x}\tilde{y})} \tag{4}$$

†If $0 \leq T < RN$, mod N can be replaced by a conditional subtraction.

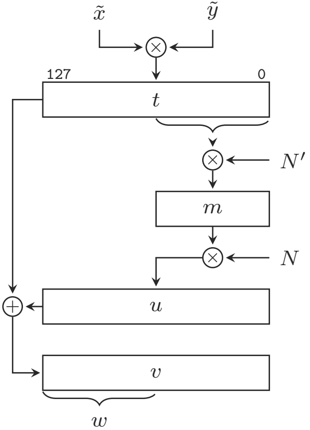
- Additions and subtractions are the same in Montgomery representation:

$$\widetilde{x \pm y} = (\tilde{x} \pm \tilde{y}) \bmod N$$

- For multiplications:

$$\widetilde{xy} = w - N\,[w \geq N]$$

  - With $R = 2^{64}$ the high and low parts of intermediate products are manipulated almost independently



⇒ **modular arithmetic loops can be efficiently vectorised with SVE**

Number-theoretic transforms (NTTs)

- A generalisation of the Discrete Fourier Transform (DFT) defined over a ring or field

$$X_k = \sum_{j=0}^{m-1} x_j g^{-jk} \bmod p \tag{5}$$

- **Operations are devoid of errors**
- Used to perform convolutions with guarantee of exact results
- Most FFT algorithms can be used with NTTs, as long as operations are recast to the new domain

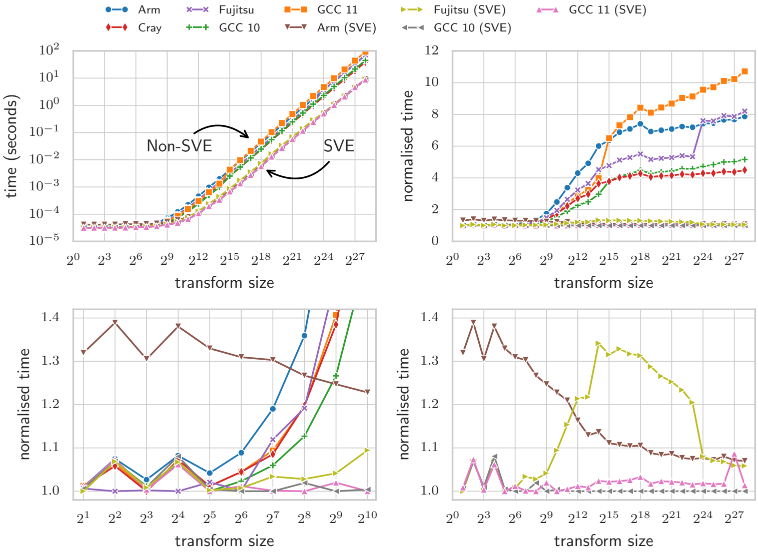
Optimising NTTs with SVE (I)

- Tests on an A64FX processor
  - **SVE** version implemented with **ACLE intrinsics**
- Major compilers
  - Arm 21.0
  - Cray 10.0.1[†]
  - Fujitsu 4.3.1,
  - Gnu 10.2 & 11

```
inline svuint64_t
mul_sve(svbool_t pg, svuint64_t a, svuint64_t b)
{
  svbool_t pc;
  svuint64_t m, xh, xl, z;

  /* x = a*b */
  xl = svmul_x(pg, a, b);
  xh = svmulh_x(pg, a, b);

  /* m = (x*N') mod R */
  m = svmul_x(pg, xl, N_prime);

  /* c = (x mod R)+(m*N mod R) >= R */
  pc = svcmpgt(pg, xl, svmla_x(pg, xl, m, N));

  /* z = (x+m*N)/R + c */
  z = svadd_x(pg, xh, svmulh_x(pg, m, N));
  z = svadd_m(pc, z, (uint64_t)1);

  /* adjust result */
  pc = svcmpge(pg, z, N);
  return svsub_m(pc, z, N);
}
```

[†] Did not support SVE intrinsics.

Optimising NTTs with SVE (II)

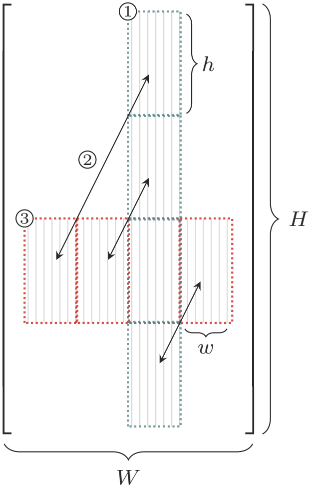


- Hybrid MPI and OpenMP implementation
- Adaptation of Bailey's four-step algorithm[1]
    0. Organise array of m points into a $H \times W$ distributed matrix
    1. Perform NTTs on the columns
    2. Apply twiddle factors
    3. Partially transpose the matrix*
    4. Perform NTTs on the rows*
- Each MPI process works on a portion of the global matrix
    - OpenMP threads do the work in parallel

[1] David H. Bailey (1990). "FFTs in external or hierarchical memory".

Distributing and parallelising NTTs (II)

1. Each process **starts with a slice of columns** $H \times w$
   - **Compute NTTs of the columns** in parallel and **apply twiddle factors**
   - Master thread **exchanges columns** with other processes
2. Once all blocks have been exchanged, **rearrange data into slice of rows**
   - Can be seen as **transposing a** $\boldsymbol{P \times w}$ **matrix**, moving h elements at a time
   - **Done in-place** by following cycles and **utilising tasks for parallelism**
3. Perform **NTTs on the rows in bulk**



Distributing and parallelising NTTs (III)

- NTTs on the columns done with Stockham algorithm[1]
  - Requires an additional buffer, but **output is in-order**
- NTTs on the rows done in bulk with DIF/DIT algorithms[1]
  - In-place algorithms (**no additional buffer required**)
  - Bit-scrambling unnecessary
  - Two possibilities for parallelisation
    1. Subdivide the rows amongst the threads
    2. **Parallelise the butterfly loops**

- Typical loop structure for DIF

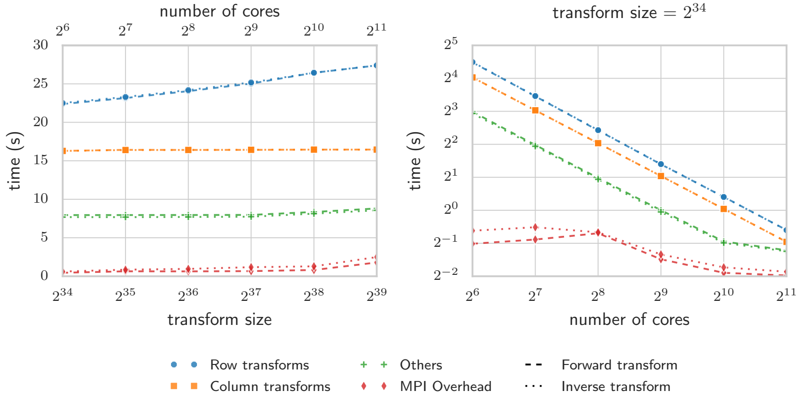
```
for(ulong k = m/2; k > 0; k >>= 1) {
  for(ulong j = 0; j < k; j++)
    for(ulong i = j; i < m; i += 2*k) {
      /* ... */
    }
}
```

- DIF with butterfly loops collapsed

```
for(ulong k = m/2; k > 0; k >>= 1) {
#pragma omp for schedule(...)
  for(ulong h = 0; h < m/2; h++) {
    ulong i = h/k, j = h&(k-1);
    /* ... */
  }
}
```

[1] Richard Crandall & Carl Pomerance, "Fast Algorithms for Large-Integer Arithmetic", in *Prime Numbers: A Computational Perspective.*

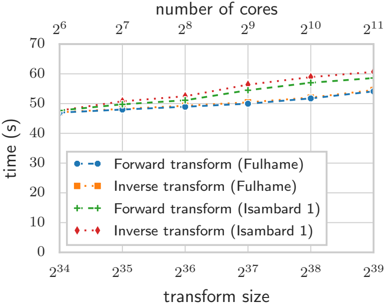
Computation breakdown of distributed NTTs †

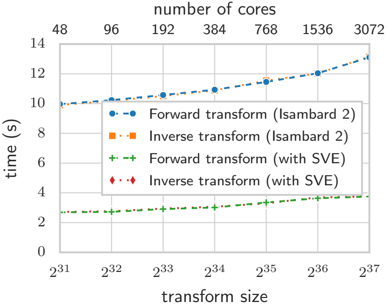


†On Fulhame (HPE Apollo 70).

Distributed and parallel NTTs on other systems

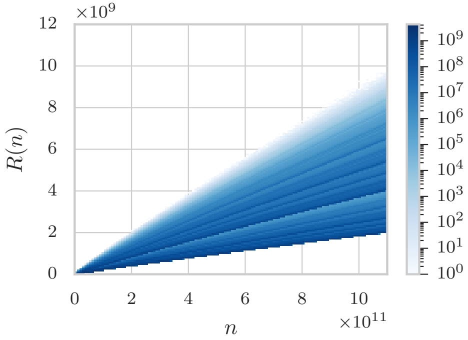
- Isambard 1 (Cray XC50) shows ∼10% slowdown wrt Fulhame
- SVE delivers over 3.5x speedup on Isambard 2 (HPE Apollo 80)





Preliminary results on Goldbach partitions

- **Counts up to 2^{40}** in less than 10 minutes
- In the near future, our **goal is to reach at least 2^{45}**
- The analysis of these results will be the subject of another publication



Conclusions

- **SVE** shows great potential **for methods that utilise modular arithmetic**
  - Easily achieved speedups greater than 4
  - Competitive even for extremely short transform sizes
- Our **NTT methods scale well** on the Arm-based systems we tested
  - **MPI almost perfectly overlapped** with computation
  - Enables **fast, efficient, and exact convolutions**
  - Underlying ideas can be applied to FFTs
- These methods have been put into action to **compute the number of Goldbach partitions** of all even numbers up to 2^{40} (**soon $\mathbf{2^{45}}$**)

Questions?

Contact: `rjj@ed.ac.uk`