REPORT ON OpenDreamKit DELIVERABLE D5.6

Parallelise the relation sieving component of the Quadratic Sieve and implement a parallel version of Block-Wiederman linear algebra over GF2 and implement large prime variants

WILLIAM HART

Due on	02/28/2017 (Month 18)
Delivered on	02/27/2017
Lead	University of Kaiserslautern (UNIKL)
Progress on and finalization of this deliverable has been tracked publicly at: `https://github.com/OpenDreamKit/OpenDreamKit/issues/119`	

CONTENTS

DELIVERABLE DESCRIPTION, AS TAKEN FROM GITHUB ISSUE #119 ON 2017-02-27

- **WP5:** High Performance Mathematical Computing
- **Lead Institution:** University of Kaiserslautern
- **Due:** 2017-02-28 (month 18)
- **Nature:** Demonstrator
- **Task:** T5.4 (#102) Singular
- **Proposal:** P. 52
- **Final report**

1. CONTEXT

One of the approaches toward OpenDreamKit's aim of High Performance Mathematical Computing is to explore the addition of very fine grained parallelism (e.g. threading or SIMD) to some key computational components like Flint and MPIR. In this deliverable, we tackle two typical algorithms of computer algebra: integer factorization and block Wiedemann for finding kernel vectors of matrices over a finite field.

Whilst the threading of the quadratic sieve looks very promising, our conclusion is that the SIMD speedup that we implemented for the block Wiedemann algorithm (and the threaded experiments we did) weren't particularly useful for the quadratic sieve. We expect that only a number field sieve implementation, with matrices that are truly massive (millions of rows) would benefit from the kind of speedup we saw.

This doesn't mean that SIMD is not useful in general, it very much is, see for example D5.5 and D5.7 where SIMD proves to be a big win. SIMD just hasn't proved to be a major benefit for a project like the quadratic sieve, where only a small part of the runtime depends on the linear algebra phase of the algorithm.

The other improvements for the quadratic sieve that we describe below, such as threading the relation sieving where we get a speedup of up to 3 on 4 cores, and the various algorithmic improvements that we describe below, turned out to be much more important in practice.

2. Report on writing a parallel implementation of the quadratic sieve

2.1. **Problem description**

The Quadratic Sieve is an algorithm for factoring integers n = pq, with n typically in the 15-90 decimal digit range. For this deliverable, we have implemented a quadratic sieve with the following features:

- Small prime variant
- Large prime variant with in-memory hash table and on-disk pseudorelation storage
- Multiple-polynomial self initialisation with Carrier-Wagstaff method
- Polynomial generation using Bradford-Monagan-Percival method
- Cache efficient sieving
- Block-Lanczos linear algebra (adapted from msieve)
- Knuth-Schroeppel multiplier
- Threaded relation sieving (using OpenMP)

2.1.1. *Results.* Some partial progress towards this goal had already been completed before the OpenDreamKit project began, but the code didn't compile or run and was in an unusable state. This included a partial implementation of a single large prime variant quadratic sieve implemented by a Google Summer of Code student, based on a prior version for small integers only. This was not parallelised and a rough sketch only. It didn't compile and wasn't correct or complete.

We have completed the implementation and the code has already been merged into the `Flint` repository.

Our implementation is not competitive below 130 bits (˜40 digits), due to the Carrier-Wagstaff/Bradford-Monagan-Percival combination. Here are some timings for larger factorisations comparing `Pari/GP` with our implementation in `Flint` on one and four cores.

In our tests, no significant improvements were observed for larger numbers of cores, so we report only on the improvements for four cores here.

n (bits)	Pari	Flint	4 core
130	0.11	0.24	0.28
140	0.22	0.33	0.30
150	0.43	0.55	0.39
160	0.79	0.99	0.56
170	1.8	1.4	0.83
180	3.8	3.0	1.7
190	8.6	4.8	2.4

200	17.6	10.2	6.0
210	38.4	20.9	13.5
220	67.7	33.2	15.8
230	199	78	34
240	257	140	52
250	444	248	92
260	1175	708	283
270	2199	1522	544

2.2. **Future possibilities**

Possible future projects could include:

- Cache efficient handling of factor base and self-initialisation data structures
- PPMPQS method
- Parallelising the file handling
- Writing an efficient self-initialising MPQS for 20-128 bits.

2.2.1. *Testing the quadratic sieve code.* The `Flint` repository is available here: `https://github.com/wbhart/flint2`

To build and test the code mentioned above, you must have `GMP/MPIR` installed on your machine (refer to your system documentation for how to do this). Then do:

```
git clone https://github.com/wbhart/flint2
cd flint
./configure --with-mpir=/path/to/mpir --with-mpfr=/path/to/mpfr --enable
export OMP_NUM_THREADS=4
make
make check MOD=qsieve
```

Full instructions on how to build `Flint` are available in the `Flint` documentation, available at the `Flint` website.

The quadratic sieve functionality is made available by including qsieve.h, and the main interface is via the function:

```
void qsieve_factor(fmpz_factor_t factors, const fmpz_t n)
```

2.3. **Blog post**

A blog post about the improvements to the quadratic sieve in Flint is available at `https://wbhart.blogspot.de/2017/02/integer-factorisation-in-flint.html`.

3. Report on parallel block Wiedemann

3.1. **Problem description**

The Block Wiedemann algorithm computes kernel vectors of a matrix over a finite field. Along with the Block Lanczos algorithm it is often used in the Quadratic Sieve and Number Field sieve over GF2, but also has other applications over a prime p.

3.2. **Work done**

We investigated parallelising the Block Wiedemann algorithm using the external library `M4RI` for basic linear algebra over GF2. This required three components to be written.

- A sparse matrix library over GFp for `Flint`
- An implementation of the Block Wiedemann algorithm over GF2
- A naive quadratic sieve to generate realistic data

These were implemented by Anders Jensen and Alex Best and are available as modules for Flint and as an external implementation respectively.

3.3. **Conclusions**

Our experiments showed that it was possible to make use of the highly optimised SIMD arithmetic in the external library `M4RI` as a form of parallelism, to speed up the Block Wiedermann algorithm, and that this outperformed Block Wiedermann using `M4RI` in threaded mode, at least for the size and sparsity of problem encountered in the quadratic

sieve! The result was comparable to the Block Lanczos code already used in Flint for the quadratic sieve.

The sparse matrix library was not the focus of this deliverable, but will be expanded at a later date and included in Flint. Because of the additional external dependency on `M4RI`, the Block Wiedemann implementation isn't eligible for inclusion in Flint, but the code is available to members of the OpenDreamKit collaboration. Future work may see the removal of this external dependency so that it can be merged into Flint. However, we currently feel that this isn't something that should be made a priority.

In fact, the quadratic sieve implementation we now have, even in parallel mode, doesn't have linear algebra as a bottleneck. This is because of the extremely small factor base sizes that are possible with the particular implementation of the quadratic sieve that we completed as part of this project.