From Multicore to Manycore Architectures: The Reduction of Parallelization Overheads and its Impact on Implementing Polynomial Arithmetic

Several classical software packages (NTL, FFTW, Spiral) implement serial algorithms fast Fourier transforms (FFT) which reaches peak performance even for input vectors of relatively small sizes, say 2^{10}. However, parallel counterparts of those algorithms require much larger sizes (say 2^{20}) in order to reach linear speedup on multicore architectures. As a consequence, some higher-level algorithms (such as those based on subproduct trees for fast polynomial arithmetic) that require parallel FFTs in order to expose parallelism, are only implemented serially on today's multicores.

In this work, we demonstrate that this dramatic observation is no longer true on graphics processing unit (GPUs). On these latter architectures parallelization overheads are much less significant since the hardware schedules threads at essentially no cost. Thanks to this feature, we have obtained attractive performances for parallel FFT-based polynomial arithmetic on GPUs.

As for serial code on CPUs, our parallel code for FFT-based polynomial arithmetic relies on base cases where plain, i.e non-fast, algorithms are employed. Parallelizing those plain algorithms is also required. We show that the long multiplication can be efficiently parallelized on GPUs. Remarkably, this latter code outperforms (highly optimized) FFT-based multiplication up to degree 2^{12} while on CPU the same threshold is only at 2^6.

In a third part of this work, we turn our attention to one of the most challenging parallelization problem in polynomial arithmetic: the Euclidean Algorithm. Indeed, there is no parallel version of this algorithm which would be both sublinear and work-efficient. The best parallel version of the Euclidean Algorithm which is work-efficient, is that for systolic arrays, (a model of computation formalized by H. T. Kung and Charles E. Leiserson in 1974) for which the span is linear. However, multiprocessors based on systolic arrays are not so common.

We report on a GPU implementation of the Euclidean Algorithm which is both work-efficient and runs in linear time for input polynomials up to degree 2^{18}. Such sizes are sufficient for many applications. Moreover, this GPU code outperforms algorithms for polynomial GCD computations that are asymptotically faster, but available only as serial CPU code, due to the same parallelization challenges as the Euclidean Algorithm.

Marc Moreno Maza
University of Western Ontario
moreno@csd.uwo.ca

.